TestStand Semiconductor Module

2023-08-18

Contents

TestStand Semiconductor Module

Use this help file, the getting started content, and the example programs to learn more about the following features and functionality the TestStand Semiconductor Module™ (TSM) adds to TestStand for developing semiconductor test system software:

- Multisite pin map file, TSM Code Module APIs, and Semiconductor Multi Test step type for developing a semiconductor test program that runs on multiple test system hardware configurations with a variable number of test sites at a high parallel test efficiency.
- Support for binning devices under test (DUTs) based on test results.
- Support for exporting and importing test limits with text files.
- Data types, configurable callbacks, and dialog boxes for specifying test program settings, test station settings, and test lot information.
- Plug-in architecture for handler and prober communication using TSM handler/prober drivers.
- Result processing plug-ins for generating report and data log files for the test lot, such as Standard Test Data Format (STDF) log file.
- Plug-in architecture for performing part average testing.
- Customizable operator interface for executing test programs, enabling or disabling test sites, displaying statistics, and configuring test lot information and test station settings.
- Customizable pin- and site-aware instrument panel VIs for debugging instruments during test program execution at a breakpoint.

Note If you open help files directly from the `<TestStand>/Doc/Help` directory, NI recommends that you open `TSHelp.chm` first because this file is a collection of all the TestStand help files and provides a complete table of contents and index.

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window.

TestStand Semiconductor Module Licensing Options

After you install TSM, you must use the NI Activation Wizard to activate the software or initiate the evaluation period for the software. When you activate TSM, you need the serial number and the name of the software kit. You can find both of these items on the Certificate of Ownership card included in your software kit.

NI offers a variety of licenses for the different ways you can use TSM in development and deployment applications. You can select from the following types of licenses: TestStand Semiconductor Module Development License and TestStand Semiconductor Module Debug Deployment Environment License.

Use the following descriptions only as a reference for the licensing options. Refer to ni.com/activate for more information about activating TestStand licenses. Contact a local NI representative for more information or for questions about specific licensing needs.

Note This document does not replace the NI Software License Agreement installed in the `<National Instruments>\Shared\MDF\Legal\license\NIReleased` directory.

- TestStand Semiconductor Module Evaluation Package
- TestStand Semiconductor Module Development License (783522-35)
- TestStand Semiconductor Module Debug Deployment Environment License (779991-35)
- TestStand Semiconductor Module Base Deployment License (784608-35)

TestStand Semiconductor Module Evaluation Package

When you run TSM in Evaluation Mode, the software expires after 7 days. The software runs as a fully functional Development System during the evaluation period.

You can use an ni.com User Profile to extend the evaluation period for an additional 45 days. You can activate a license at any point during or after the evaluation period.

TestStand Semiconductor Module Development License (783522-35)

Activate the TestStand Semiconductor Module Development System License to develop and edit test programs within the TestStand Sequence Editor. You must have an ni.com User Profile to activate an TestStand Semiconductor Module Development System License.

TestStand Semiconductor Module Debug Deployment Environment License (779991-35)

The TestStand Semiconductor Module Debug Deployment Environment License offers the most flexibility for deploying TestStand, LabVIEW-based, Measurement Studio-based systems.

Activate this license to install the development versions of TestStand, LabVIEW, Measurement Studio, Switch Executive, TSM, and any corresponding add-on toolkits on a single test station so you can debug deployed test applications on the test station. This license grants the ability to make minor edits to fix bugs in deployed test applications but does not grant the ability to perform any development tasks using TestStand, LabVIEW, or Measurement Studio on the test station.

You cannot activate and deactivate the TestStand Semiconductor Module Debug Deployment Environment License and reuse it on multiple computers. If you need to use a single debug license across multiple computers, contact NI for more information about the Concurrent TestStand Semiconductor Module Debug Deployment Environment License.

TestStand Semiconductor Module Base Deployment License (784608-35)

The TestStand Semiconductor Module Base Deployment License is the minimum license required for all deployed TSM-based applications. Activate this license to deploy the TSM Runtime, the TestStand Runtime, and the Switch Executive Runtime. The Base Deployment License enables you to run a TSM operator interface and test programs on the single test station to which the license applies. This license does

not grant the ability to perform any development tasks using the TestStand Sequence Editor, a TestStand custom sequence editor, or the TestStand API.

What's New in TSM

Each new version of TSM includes new features, API additions and changes, and compatibility and known issues.

What's New in the TestStand 2021 Semiconductor Module 2023 Q1

The TestStand 2021 Semiconductor Module 2023 Q1 introduces new features.

TestStand 2021 Semiconductor Module 2023 Q1 New Features

Learn what's new in TestStand 2021 Semiconductor Module 2023 Q1.

- Use the TSM Runtime Data Viewer to see test results and debug issues at runtime.
- See which alarms were raised during the execution of a test program in the Lot Summary Report.
- When used with STS Software 23.0 and later, TSM uses the STS blind-mate port names for RF instruments instead of the instrument driver port names. TSM displays the blind-mate port names in the Pin Map Editor and returns the blind-mate port names as the channel names for RF pin queries.

What's New in the TestStand 2021 Semiconductor Module 2022 Q2

The TestStand 2021 Semiconductor Module 2022 Q2 introduces new features. Some compatibility issues might exist as a result of changes in the TestStand 2021 Semiconductor Module 2022 Q2.

TestStand 2021 Semiconductor Module 2022 Q2 New Features

Learn what's new in TestStand 2021 Semiconductor Module 2022 Q2.

- Run TSM with TestStand 2021.

- Use LabVIEW 2021 to run TSM examples and tutorials and develop TSM code modules.
- Quickly debug Semiconductor Sequence Call steps by reviewing test results on the Tests tab at runtime.
- Get and set sessions through the Model-Based Instruments API.
- Write code modules with the Mixed-Signal TSM Python API.

TestStand 2021 Semiconductor Module 2022 Q2 Compatibility

Refer to the TestStand 2021 Semiconductor Module 2022 Q2 Release Notes on the NI website for a list of known issues in TestStand 2021 Semiconductor Module 2022 Q2.

The TestStand 2021 Semiconductor Module 2022 Q2 introduces the following behavior changes between version 2021 Q4 and version 2022 Q2:

- TSM no longer includes step type templates. TSM continues to provide the mechanism for other add-ons to install step type templates.
- TSM drops support for LabVIEW 2017. TSM examples and tutorials that use LabVIEW are no longer located within year-specific folders. For example, the Accelerometer with LabVIEW 2018 example is now located in `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\Accelerometer.seq` instead of in `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\2018\Accelerometer.seq`.

What's New in the TestStand 2020 Semiconductor Module 2021 Q4

The TestStand 2020 Semiconductor Module 2021 Q4 introduces new features. Some compatibility issues might exist as a result of changes in the TestStand 2020 Semiconductor Module 2021 Q4.

TestStand 2020 Semiconductor Module 2021 Q4 New Features

Learn what's new in TestStand 2020 Semiconductor Module 2021 Q4.

- Use the Perform Part Average Testing step to perform part average testing for any tests with part average testing enabled that have already been performed for the current part. Use the Perform Part Average Testing step to

determine whether to perform part average testing before or after inline QA and other steps.

- Call larger and more complex test sequences with the Semiconductor Sequence Call step. Use the Semiconductor Sequence Call step's Step Name.Published Data Id field to assign tests to Semiconductor Multi Test steps, even if multiple Semiconductor Multi Test steps in the called sequence publish the same data ID.
- Analyze sequences that include the Semiconductor Sequence Call step in the TestStand Sequence Analyzer to resolve errors and warnings before deploying your test program.
- Customize the MIR EXEC_TYPE field in your STDF records to simplify downstream STDF tools and scripts.

TestStand 2020 Semiconductor Module 2021 Q4 Compatibility and Known Issues

Refer to the TestStand 2020 Semiconductor Module 2021 Q4 Release Notes on the NI website for a list of known issues in TestStand 2020 Semiconductor Module 2021 Q4.

The TestStand 2020 Semiconductor Module 2021 Q4 introduces the following behavior changes between version 2020 and version 2021 Q4:

- The default value of the Master Information Record (MIR) `EXEC_TYPE` field has changed to `NI STS Software` and is no longer dependent on the version of the STS Software you have installed. You can also customize the `EXEC_TYPE` field value.

TestStand 2020 Semiconductor Module 2021 Q4 Bug Fixes

Refer to the TestStand 2020 Semiconductor Module 2021 Q4 Release Notes on the NI website for a list of bugs fixed in TestStand 2020 Semiconductor Module 2021 Q4.

This is not an exhaustive list of issues fixed in TestStand 2020 Semiconductor Module 2021 Q4.

What's New in the NI TestStand 2020 Semiconductor Module

The NI TestStand 2020 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2020 Semiconductor Module.

NI TestStand 2020 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2020 Semiconductor Module (TSM) and other changes since the NI TestStand 2019 Semiconductor Module.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Alarms

The alarms feature allows NI instrument drivers to report error conditions for an instrument at run-time. TSM checks for alarms after each test step and allows you to specify how alarms are handled on a per-alarm and per-pin basis.

Note NI TestStand 2020 Semiconductor Module supports the `ComplianceAlarm` for NI-DCPower 20.1 and later. The `ComplianceAlarm` indicates that the instrument state was not at its programmed value or exceeded programmed limits when a measurement was made, which invalidates the measurement.

Grouping NI-DCPower Instrument Channels for Use in Multi-Channel Sessions

You can group multiple NI-DCPower instrument channels and treat them as a single logical instrument and control them in one session. When all NI-DCPower instrument channels belong to a single group you can avoid using session loops in code modules. The instrument driver performs most operations on multiple channels in a single function call in parallel to achieve improved multisite efficiency. Refer to the instrument driver help for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

By default, when you create a new NI-DCPower instrument in the pin map file, TSM

creates a single channel group containing all instrument channels. TSM creates a single session for each channel group of NI-DCPower instruments in the pin map file. TSM 2019 and earlier do not allow for channel grouping and pin maps created with older versions of TSM do not contain channel group information. You must convert all NI-DCPower instruments in these pin maps to use channel groups.

Offline Mode

The default main menu now displays an **Offline Mode** indicator to the right of the **Help** menu when you enable Offline Mode. When you disable Offline Mode, the default main menu no longer displays the **Offline Mode** indicator.

Semiconductor Sequence Call Step

Use the Semiconductor Sequence Call step to call a sequence and pass tests to Semiconductor Multi Test steps in the called sequence.

Set Relays Step

Use the Set Relays step to control relays and to apply relay configurations.

Test Program Performance Analyzer

- You can add notes and additional metadata to Test Program Performance Measurement Data Logs for improved record keeping during performance optimization.
- Use the Log Browser Window to specify a directory of performance log files, view log files and their metadata, and select log files to open and/or compare.
  - For improved performance while loading data in the in the Test Program Performance Analyzer, you can load only a sample of log files within a directory by specifying the **Log Data Sample Rate**.

Additional Improvements

The NI TestStand 2020 Semiconductor Module includes the following additional enhancements:

- Use Semiconductor Multi Test steps and Semiconductor Action steps in most types of loops without the possibility of incorrect results. There remain

some limitations to using TSM steps in loops but the steps report run-time errors in those situations instead of producing incorrect results.

- Specify the relays you use in your code module on the Options tab of a Semiconductor Multi Test or Semiconductor Action step that uses the code module.
- New TSM sequence analyzer rules return errors in the following situations:
  - When instruments defined in the pin map are missing from Measurement & Automation Explorer (MAX)
  - When the **Specify DUT Pins** or **Specify Site Relays** option on the Options tab of a Semiconductor Multi Test step or a Semiconductor Action step is enabled, but some of the included DUT pins or relays are not in the pin map file.
- Create a basic TSM test program from a digital pattern project. Select **Semiconductor Module»Create Test Program from Digital Pattern Project** to launch the Create Test Program from Digital Pattern Project dialog box.
- Use an expression to determine the software bin at run time. The Result.Evaluations property of the Semiconductor Multi Test step type includes a new `FailBinExpr` property, which is an expression that determines the software bin at run time. If the test fails and this expression is not empty, the Semiconductor Multi Test step evaluates the expression and copies the evaluated value to the `FailBin` property.
- Support for logging failed cycle information from NI-Digital Pattern Drivers in .NET applications.
- The Create Multisite Data for Analog Output VI and corresponding .NET method have been updated. The Create Multisite Data for Analog Output VI for DAQmx now supports multiple task pin queries and the **Site Pin Data** indicator for the Per Site Data instance of this VI has improved data representation.
  Outputs from the TSM .NET API more closely match the DAQmx API and the `samplesPerSitePerPin` parameter of the `CreatePerSiteMultisiteDataForDAQmxAnalogOutput` .NET method, which supplants the `Create`

`PerSiteMultisiteDataForAnalogOutput` .NET method, has improved data representation.

NI TestStand 2020 Semiconductor Module Compatibility and Known Issues

Refer to the List of Known Issues in NI TestStand 2020 Semiconductor Module on the NI website for a list of known issues in NI TestStand 2020 Semiconductor Module (TSM).

The NI TestStand 2020 Semiconductor Module introduces the following behavior changes between version 2019 and version 2020:

- You cannot use multiple Semiconductor Multi Test steps or Semiconductor Action steps configured to use multiple threads in While loops, in Do While loops or in For loops that use the Custom Loop option when performing multisite testing. The steps report a run-time error in these situations. Use other types of loops instead, such as For loops that use the Fixed Number of Iterations option.

  Previously, NI recommended not to use Semiconductor Multi Test steps or Semiconductor Action steps in loop blocks that the TestStand For step and For Each step create because using multiple Semiconductor Multi Test steps or Semiconductor Action steps in a loop can result in incorrect step results in certain multisite situations. NI previously recommended making the last step in the loop block a Semiconductor Multi Test step or a Semiconductor Action step with the **Multisite Option** set to `One thread only` to avoid incorrect behavior of Semiconductor Multi Test steps or Semiconductor Action steps in a loop.
- For code modules that use relays, you must specify the relays on the Options tab of the Semiconductor Multi Test step or Semiconductor Action step that uses the code module.

  Previously, TSM included all relays in the `SemiconductorModuleContext` for all steps. As a result, a test step that uses relays and executes successfully in previous versions of TSM might generate a run-time error in

version 2020. To prevent the run-time error, specify the relays that the code module uses on the Options tab.

NI TestStand 2020 Semiconductor Module Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2020 Semiconductor Module on the NI website for a list of bugs fixed in NI TestStand 2020 Semiconductor Module.

This is not an exhaustive list of issues fixed in NI TestStand 2020 Semiconductor Module.

What's New in the NI TestStand 2019 Semiconductor Module

The NI TestStand 2019 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2019 Semiconductor Module.

NI TestStand 2019 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2019 Semiconductor Module (TSM) and other changes since the NI TestStand 2017 Semiconductor Module.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Offline Mode

Use Offline Mode in TSM to develop, run, and debug test programs only on a computer without access to NI instruments. Ensure you meet the requirements on the computer on which you want to use Offline Mode.

Grouping Instruments for Use in Multi-Instrument Sessions

You can group multiple NI-Digital Pattern instruments or multiple NI-SCOPE instruments together and treat them as a single instrument. When all the instruments of the same type belong to a single group or when the instruments of the same type in a subsystem belong to a single group, you do not need to use parallel For Loops to iterate over the instrument driver sessions. The instrument driver specific to the grouped instruments performs most operations on all channels

in parallel to achieve improved multisite efficiency. Refer to the instrument driver help for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

By default, when you create a new NI-Digital Pattern instrument or a new NI-SCOPE instrument in the pin map file, TSM sets the `group` attribute to `Digital` or to `Scope` so that all newly created NI-Digital Pattern instruments or NI-SCOPE instruments belong to the same group. TSM creates a single session for each group of NI-Digital Pattern instruments in the pin map file. TSM 2017 and earlier do not automatically group NI-Digital Pattern instruments or NI-SCOPE instruments together in pin map files. Use the Pin Map Editor to modify existing pin map files to change the value of the **Group** option for each instrument to assign the instrument to the same group.

Instrument Model Library

The Instrument Model Library is a collection of XML files that describe instruments. Instrument model description files include general information, details for connection components (channels, ports, resources), and configuration properties of the instrument and its resources. The instrument model description file does not include session information.

Use the Instruments tab of the Pin Map Editor to add and modify instances of model-based instruments in the pin map.

Use the following new TSM Code Module API VIs to return the names, properties, and values of model-based instruments:

- **Get All Model-Based Instrument Names**—Use this VI to return the instrument names and models for all model-based instruments in the Semiconductor Module context. You can use instrument names to query the model properties for the information needed to create the appropriate sessions to drive the instrument.
- **Get Model-Based Instrument Property List**—Use this VI to return an object containing the name of the model and an array of ModelBasedInstrumentProperty objects that contain the names and values of the instrument properties.
- **Get Model-Based Instrument Resource Property List**—Use this VI to return an array of IModelBasedInstrumentResourcePropertyList objects where

each element contains the name of a resource as well as an array of ModelBasedInstrumentProperty objects that contain the names and values of the instrument resource properties.

- **Get Model-Based Instrument Property Value**—Use this VI to return the value of a named property and a Boolean value indicating whether or not the named property was found in the property list supplied.
- **Get Model-Based Instrument Resource Property Value**—Use this VI to return the value of a named property from a named resource and a Boolean value indicating whether or not the named property was found in the array of resource property lists supplied.

Connecting Shared Resources in a Pin Map

- Use the Connections table to connect shared resources in a pin map. A shared resource is a device on the tester or DIB that is connected to an instrument or relay driver module and shared by multiple sites. You can connect shared resources using system pins and system relays, or by using DUT pins and site relays.

Relay Configurations

- In the Pin Map Editor, you can now create relay configurations to set multiple relays to a state defined in the pin map.
- Use the Apply Relay Configuration VI or the `ApplyRelayConfiguration` .NET method to perform relay actions on the relays in the relay configuration.

CSV Test Results Log

You can now enable and configure the new CSV Test Results Log to store data in a comma-separated values text file, which provides better performance than the Debug Test Results Log result processor in a production environment. The CSV Test Results Log result processor generates a single file for all sites in the test program. You can open the `.csv` file directly in a spreadsheet application for analysis or to correlate test results.

The Test Results Log is now the Debug Test Results Log.

InstrumentStudio Integration

- Use the InstrumentStudio toolbar button to launch InstrumentStudio, which is a pin- and site-aware, software-based front panel application you can use to monitor, control, and record measurements from supported devices.
- You can also use the InstrumentStudio Project Panel in the Test Program Editor to launch InstrumentStudio, or to specify an InstrumentStudio configuration file to use in a test program.

Note If you launch InstrumentStudio in any other way, such as from the Microsoft Windows Start menu, InstrumentStudio is not pin and site aware.

Multisite Scaling Improvements

TSM 2019 includes extensive optimizations for multisite testing with improved parallel test efficiency (PTE). These performance improvements can significantly increase throughput for systems, depending heavily on the specifications of the system.

Test Program Development and Debugging Improvements

- You can now configure TSM-specific environments. TSM is enabled by default in custom TSM environments you create. You can disable and re-enable TSM in a custom TSM environment. Use the Configure Environment dialog box to create, load, and edit an environment.
- You can now use the Site Data Exists and the Global Data Exists TSM Code Module API to check for the existence of per-site or global data.
- Switching settings for TSM Step Types are disabled. Use relays in the pin map to perform switching operations.
- The Test Program Performance Analyzer graphs include the following changes:
  - The Average Step Times graph is now the Step Time Statistics graph with options to show average step times, PTE%, and a distribution box plot.
  - You can now view statistical distribution of test times as a scatter distribution.

- You can log failed cycle information from the NI-Digital Pattern Driver to STDF Log files.
- Use the Per-Instrument to Per-Site Pattern Results VI to convert Pass/Fail pattern burst results from a call to the NI-Digital Pattern driver into per-site results that you can use in per-site loops in code modules.
- Use the Per-Instrument to Per-Site Data VI or `PerInstrumentToPerSiteData` .NET method to transform data that is arranged by instruments and channels into data arranged by sites and pins.
- The LabVIEW VI Analyzer now verifies that VIs use the High Resolution Polling Waiting VI instead of other wait functions to improve wait time precision.
- TSM now supports pin-based per-site publishing. Use the optional **Pin** and **Published Data Id** parameters of the Publish Data VI or `PublishPerSite` .NET method to publish data for each site to tests that have non-empty **Pin** and **Published Data Id** fields.
- You can now use the Publish Data VI to publish data for a single value from a single site.
- In LabVIEW, you can now use the TSM controls instead of string controls to select from the pins, relays, relay configurations, specifications, published data IDs, and input data IDs of a linked sequence file. To use this feature, you must link a LabVIEW project file to a TSM sequence file.
- Updated step templates better conform to suggested NI code style guidelines and test program structure. The step templates include the following major changes:

Type of Change	Details
Source File/Directory Structure	▪ LabVIEW version-specific directories ▪ Template-specific directories ▪ `Template_` prefix changed to `Template - .`
Behavior of Adding to Test Program	▪ Template-specific default location ▪ Instantiating a .NET template contained within a .NET assembly with the same na

	me as a template that has already been in stantiated automatically references the ex isting assembly
Significant Coding Style Changes	▪ Use of the High Resolution Polling Wait VI for increased precision in wait times ▪ Use of re-entrant VIs ▪ Disabled automatic error handling ▪ VIs no longer contain compiled code ▪ Changed connector panes ▪ Eliminated coercion dots by explicitly c onverting data ▪ Renamed error and TSM context control s and indicators

Additional Enhancements

Additional Improvements

The NI TestStand 2019 Semiconductor Module includes the following additional enhancements:

- The STDF Log now includes the correlation offsets file path you specify in the Load Correlation Offsets step as a DTR.
- In the Pin Map Editor, you can now change the type of an instrument and maintain the relevant property values and existing connections among pins, relays, instruments, and channels. Right-click the instrument and select **Change to** in the context menu or use the **Instrument Type** drop-down menu to select the new instrument type.
- The TSM Operator interface now displays the active STS Software installed on the system.
- The Pin(s) to NI-Digital Pattern Session(s) VI and the `GetNIDigitalPatternSession(s)` .NET methods now support pin queries for single or multiple pins that return a single NI-Digital instrument.
- The Set Site Data VI is now a polymorphic VI that allows the use of 2D arrays, in which each row of the array contains data corresponding to one site.

- You can now use the generic versions of the `GetSiteData` and `GetGlobalData` .NET methods to specify a data type to cast the retrieved data to before returning it.
- If you configure the LabVIEW Adapter to use the LabVIEW Development System, the window title of the VI clone displays a comma-separated list of the site numbers executing on that VI clone. For example, the window title for a VI clone that executes sites 1 and 2 for the Continuity VI will read: Continuity.vi: Site(s) 1,2.
- The Lot Summary Report now includes a Test Results section that contains a table of test evaluation results by site for all the tests that executed at least once in the lot, sorted by execution order. You can use this data to compare the results of lots, which can be helpful during debugging. You can programmatically obtain the same data with the Test Results Statistics TSM Application API.
- The Test Results Log is now the Debug Test Results Log. With the new performance improvements to the Debug Test Results Log, you can now leave the Debug Test Results Log enabled during production. Enabling the Debug Test Results Log might affect the performance of a test program. Perform a benchmark to ensure no performance loss exists when you enable the Debug Test Results Log.

NI TestStand 2019 Semiconductor Module Compatibility and Known Issues

Refer to the List of Known Issues in NI TestStand 2019 Semiconductor Module on the NI website for a list of known issues in NI TestStand 2019 Semiconductor Module (TSM).

The NI TestStand 2019 Semiconductor Module introduces the following behavior changes between version 2017 and version 2019:

- TSM 2019 modifies the behavior of the Control Relay VI Multiple Relays - Multiple Actions polymorphic instance and the `ControlRelay` .NET method. Final relay state positions are now determined sequentially, based on the order of the input relays or relay groups. The following table demonstrates the change in relay behavior between TSM 2017 and TSM 2019:

Relays/Relay Groups (Input)	States (Input)	Final States (TSM 2017)	Final States (TSM 2019)
Rly1, Rly2, Rly3, Rly1	Open Relay, Close Relay, Open Relay, Close Relay	Rly1:Closed, Rly2:Closed, Rly3:Open	Rly1:Closed, Rly2:Closed, Rly3:Open
Rly1, Rly2, Rly3, Rly1	Close Relay, Close Relay, Open Relay, Open Relay	Rly1:Closed, Rly2:Closed, Rly3:Open	Rly1:Open, Rly2:Closed, Rly3:Open
Grp1 (Rly1, Rly2, Rly3), Rly1	Close Relay, Open Relay	Rly1: Closed, Rly2: Closed, Rly3: Closed	Rly1: Open, Rly2: Closed, Rly3: Closed
Rly1, Grp1 (Rly1, Rly2, Rly3)	Close Relay, Open Relay	Rly1: Closed, Rly2: Open, Rly3: Open	Rly1: Open, Rly2: Open, Rly3: Open

- You can group multiple NI-Digital Pattern instruments or multiple NI-SCOPE instruments together and treat them as a single instrument. When all the instruments of the same type belong to a single group or when the instruments of the same type in a subsystem belong to a single group, you do not need to use parallel For Loops to iterate over the instrument driver sessions. The instrument driver specific to the grouped instruments performs most operations on all channels in parallel to achieve improved multisite efficiency. Refer to the instrument driver help for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

  By default, when you create a new NI-Digital Pattern instrument or a new NI-SCOPE instrument in the pin map file, TSM sets the `group` attribute to `Digital` or to `Scope` so that all newly created NI-Digital Pattern instruments or NI-SCOPE instruments belong to the same group. TSM creates a single session for each group of NI-Digital Pattern instruments in the pin map file. TSM 2017 and earlier do not automatically group NI-Digital Pattern instruments or NI-SCOPE instruments together in pin map files. Use the Pin Map Editor to modify existing pin map files to change the value of the **Group** option for each instrument to assign the instrument to the same group.
- The Test Results Log is now the Debug Test Results Log.
- The following table lists how TSM 2019 changes the error reporting behavior when you enable or disable the STDF Log.

Conditions	STDF Log Enabled	STDF Log Disabled
Multiple tests with 0 as a test number or blank test numbers and different test names or evaluation types	Error in TSM 2017 No error in TSM 2019	No error in TSM 2017 No error in TSM 2019
Multiple tests with non-zero test numbers and different test names or evaluation types	Error in TSM 2017 Error in TSM 2019	No error in TSM 2017 Error in TSM 2019

- The Get Site Data VI and the `GetSiteData` .NET method now return an error if the site data does not exist for all the sites in the Semiconductor Module Context.
- TSM 2019 stores the path of the simulated handler driver in the `HandlerDriverSequenceFilePath` station setting property when you enable the NI built-in simulated handler driver. TSM 2017 or earlier leaves that property unchanged when you enable the simulated handler driver.
- Names for NI instruments in the pin map file are no longer case sensitive. Names for custom instruments in the pin map file remain case sensitive.
- TSM 2019 renames the **Limit Number of Results Reported** option in the Debug Test Results Log Options dialog box to **Limit Number of Results Displayed in Report View**. The option no longer applies to the Debug Test Results Log file and now applies only to the Report View.
- TSM 2019 updates the directories for some LabVIEW examples and tutorials. Verify the example and tutorial directories before using them.
- If you modified any TSM step templates, compare the modified versions to the updated versions of the step templates in NI TestStand 2019 Semiconductor Module.

NI TestStand 2019 Semiconductor Module Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2019 Semiconductor Module on the NI website for a list of bugs fixed in NI TestStand 2019 Semiconductor Module.

This list is not exhaustive. If you reported an issue to NI and received a CAR ID to track the issue, you can search for the CAR ID in the ID or Legacy ID column to determine whether the issue has been fixed.

What's New in the NI TestStand 2017 Semiconductor Module

The NI TestStand 2017 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2017 Semiconductor Module.

NI TestStand 2017 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2017 Semiconductor Module (TSM) and other changes since the NI TestStand 2016 SP1 Semiconductor Module.

 Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Improved Test Program Development

Step Templates

TSM includes the following new step types with template code:

- NI-DAQmx Create AI Voltage Tasks
- NI-DAQmx Acquire AI Voltage Waveforms
- NI-DAQmx Clear Tasks

Additional Instrument Support

The TSM Code Module API now includes support for the PXI-2567 relay driver module. Install NI-SWITCH 17.0 or later to use the built-in support for the PXI-2567 relay driver module in TSM.

Use the Relay Driver Module section of the System View in the Digital Pattern Editor 18.0 to control and monitor relays.

TSM Sequence Editor UI Configuration

The default TSM UI Configuration of the TestStand Sequence Editor includes the following changes to streamline semiconductor test program development.

- Simplified toolbar, including the following changes:

Toolbar	Change
Standard	No change
Debug	Removed
Environment	The following items have been removed: ▪ Selected Adapter ▪ User Manager ▪ Find Previous ▪ Find Next The Lock/Unlock UI Configuration toggle button is set to Lock.
Navigation	No change
Help	The following items have been removed: ▪ Guide to Documentation ▪ Getting Started ▪ Web Resources ▪ Discussion Forum
Sequence Hierarchy	Removed
Sequence Analyzer	No change
Semiconductor Module	The following debug items have been added: ▪ Step Into ▪ Step Over ▪ Step Out The following button has been added: ▪ Launch InstrumentStudio

- Altered **Execute** menu that replaces the Test UUTs and Single Pass items with the Start Lot and Single Test items
- Modified Insertion Palette pane that displays the Semiconductor Module folder and the Action Step at the top of the Step Types list

- More detailed Steps pane, including the following changes:
  - The Steps pane of the Sequence File window expands the **Description** column to include the VI name or ClassName:MethodName for the associated code module. The pane also includes new **Num Tests**, **Pins**, and **Multisite Option** columns.
  - The Steps pane of the Execution window includes a new **Module Time** column.

When you launch TSM for the first time or enable TSM, it loads the TSM UI Configuration, named `NI_SemiconductorModule`, and saves the most recently active UI configuration as `NI_SemiconductorModule_SavedLayout`. When you disable TSM, TestStand loads the `NI_SemiconductorModule_SavedLayout` UI configuration. You can modify the TSM UI Configuration and restore it to the default state.

Test Program Performance Analysis

During the test program development phase, you can now use built-in TSM tools to measure test program performance and then analyze the resulting data with the Test Program Performance Analyzer. Some common use cases include identifying the slowest test times, identifying low parallel test efficiency (PTE) values, and displaying the overall socket time and the calculated PTE value for each site configuration.

Static Code Analysis

TSM now includes general, performance, best practices, and statistics rules to use in the TestStand Sequence Analyzer. The TSM rules are enabled by default.

TSM now installs the following test, enabled by default, to the LabVIEW VI Analyzer in the **TestStand Semiconductor Module** section:

- **TSM Context Closing**—Verifies that a VI properly closes Semiconductor Module Context references. Detects cases where the output of a Semiconductor Module Context reference is not wired or not wired to a Close Reference function. Closing references in LabVIEW frees up memory that LabVIEW allocates for the references. Failure to close references causes

reference leaks, which can negatively affect the performance of the VI over time.

Refer to the **LabVIEW Help** for more information about the LabVIEW VI Analyzer. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

Enabling Sites

The default Configure Lot Settings dialog box now sets the number of sites to the number of sites in the test program pin map. You can use the **Enabled Sites** control in the dialog box to enable or disable specific sites. You can no longer use the Configure Station Settings dialog box to specify the number of sites to test.

Examples, Tutorials, and Step Templates Support LabVIEW 2018 Features

The TSM examples, tutorials, and step templates incorporate LabVIEW 2018 support for Parallel For Loops with error registers and the High Resolution Polling Wait VI. Refer to the **LabVIEW Help** for more information about these LabVIEW features. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

The examples and tutorials include `LabVIEW 2017 and earlier` and `LabVIEW 2018 and later` directories for support files based on the version of LabVIEW you want to use.

When you use TSM step templates, TSM selects the version of the VI template that corresponds to the active version of LabVIEW on the computer. You can save version-specific or version-neutral custom templates in the `<TestStand Public>` directory.

Correlation Offsets Support

Support for Correlation Offsets

You can generate a tab-delimited correlation offsets template file based on the numerical limit tests in a sequence file by selecting **Semiconductor Module»Export Correlation Offset Template file based on <filename>**. You can use the template file as a starting point for a custom correlation offsets file.

Use the Load Correlation Offsets Step and associated edit tab to load and apply correlation offset values to test results on a per-site basis at run time before

evaluating the test result data against limits. The Test Results Log includes the correlation offset values.

STS Test Head Integration

Get Test Information Step

The Get Test Settings step was redesigned and renamed to Get Test Information. Use the step and the associated edit tab to more easily obtain the values for lot settings, station settings, STS state, execution data, and custom test conditions.

Control STS Test Head Step

Use the Control STS Test Head step and associated edit tab to control properties of the STS. The step requires STS Maintenance Software 17.1 or later and must be run on an STS. Use the Get Test Information step to obtain the `TestHead.TestHeadAvailable` property at run time to determine whether you can access the STS properties.

Additional Enhancements

Additional Improvements

The NI TestStand 2017 Semiconductor Module includes the following additional enhancements:

- **Custom execution captions**—The Windows pane and the Execution window now display the site number and part ID in addition to the sequence filename and testing state.
- **Pin Group and Relay Group API**—Use the Get Pins in Pin Group(s) API to obtain a list of pins contained in the pin group or list of pin groups you specify. Use the Get Relays in Relay Group(s) API to obtain a list of relays contained in the relay group or list of relay groups you specify.
- **Numerical order of site numbers**—The Get Site Numbers VI and the `SiteNumbers` .NET property now return site numbers in numerical order instead of in a random order determined by the order in which sites execute.
- **OnSiteTestingComplete**—Use the `OnSiteTestingComplete` callback sequence to perform actions on a DUT or on instruments after all

DUT tests have completed and TSM has assigned a bin to the DUT. TSM calls the sequence after the `MainSequence` sequence, after all PAT tests complete, and after TSM assigns a bin to the DUT.

- **Report Orientation of Test Results Log**—You can now specify the orientation of the Test Results Log. The default is portrait orientation. Landscape orientation uses wider columns for tests with long test numbers or test names.
- **SemiconductorModuleManager parameter in handler/prober driver entry points**—The handler/prober driver entry points now include a **SemiconductorModuleManager** parameter, which is an object reference to an instance of a Semiconductor Module Manager that you can use in applications that use the TSM Application API. Use this object reference to get information about test execution, obtain test statistics, monitor the state of the test system, and so on.
- **Menu and Toolbar Improvements**—You can now use the Semiconductor Module menu or the TSM toolbar to launch InstrumentStudio.
- **Operator Interface Improvements**—The default TSM operator interfaces now include an **Open STS Maintenance Software** button to launch STS Maintenance Software 17.1 or later. The operator interfaces also display the status of an STS running STS Maintenance Software 18.0 or later. The operator interfaces disable the **Start Lot** and **Single Test** buttons when STS Maintenance Software 18.0 or later is using the tester.
- **Adding text data to Test Results Log**—You can now add data that is not a measurement or test limit to the Test Results Log.

See Also

NI TestStand 2017 Semiconductor Module Compatibility Issues

NI TestStand 2017 Semiconductor Module Compatibility and Known Issues

Refer to the List of Known Issues in NI TestStand 2017 Semiconductor Module on the NI website for a list of known issues in NI TestStand 2017 Semiconductor Module (TSM).

The NI TestStand 2017 Semiconductor Module introduces the following behavior changes between version 2016 SP1 and version 2017:

- TSM removed support for custom operator interfaces based on the default NI TestStand 2013 Semiconductor Module LabVIEW operator interface. Those operator interfaces might not function correctly in NI TestStand 2017 Semiconductor Module. NI TestStand 2014 Semiconductor Module introduced the TSM Application API and significant changes to the default operator interfaces to simplify operator interface implementation. You must upgrade custom operator interfaces to use this new technology.
- TSM removed many custom properties from the `NI.SemiconductorModule` container in the attributes of the `ModelData` container passed to process model plug-ins. Previous versions of TSM used these properties to communicate information among TSM process model plug-ins. This information is now available only in the TSM Application API and the Get Test Information step.
- TSM 2017 installs `SeqEdit.exe.config` in the `<TestStand>\Bin` directory and overwrites any existing file with the same filename in that directory. If you have created a custom `SeqEdit.exe.config` file, complete the following steps to preserve the custom file.

  1. Move the existing `SeqEdit.exe.config` file to a location outside the `<TestStand>` directory.
  2. Install TSM 2017.
  3. Compare the `SeqEdit.exe.config` file TSM 2017 installed to the custom version of the file and merge the custom changes into the file TSM 2017 installed.

NI TestStand 2017 Semiconductor Module Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2017 Semiconductor Module on the NI website for a list of bugs fixed in NI TestStand 2017 Semiconductor Module.

This list is not exhaustive. If you reported an issue to NI and received a CAR ID to track the issue, you can search for the CAR ID in the ID or Legacy ID column to determine whether the issue has been fixed.

What's New in the NI TestStand 2016 SP1 Semiconductor Module

The NI TestStand 2016 SP1 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2016 SP1 Semiconductor Module.

NI TestStand 2016 SP1 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2016 SP1 Semiconductor Module (TSM) and other changes since the NI TestStand 2016 Semiconductor Module.

 Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Improved Test Program Development

Step Templates

Use the TSM step types with template code to perform common operations, such as setting up and closing instruments, powering up a DUT, or executing common tests. You can modify the code to customize the behavior of the step within a test program.

TSM provides template code for the following step types:

- Setup and Close
  - NI-DCPower Close
  - NI-DCPower Initialize
  - NI-Digital Pattern Close
  - NI-Digital Pattern Initialize
- DUT Power Up
- DUT Power Down
- Burst Pattern
- Continuity Test

- Leakage Test

Additional Instrument Support

The TSM Code Module API now includes implementations for NI-DAQmx, NI-DMM, NI-FGEN, and NI-SCOPE.

Custom Instrument Panels

You can now create custom pin- and site-aware instrument panel VIs to debug instruments during test program execution at a breakpoint, which can be useful during test program development and troubleshooting. Custom instrument panels obtain active instrument sessions stored in Semiconductor Module context objects using the Set Session VIs or .NET methods in the TSM Code Module API. During active test program execution, TSM disables the custom instrument panel to avoid conflicts.

The custom instrument panel components must reside in the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\InstrumentPanels` directory for the panels to appear in the Semiconductor Module menu.

Refer to the Parametric I/V Instrument Panel example and the following resources that you can use as starting points for custom instrument panels you create:

- Examples located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instrument Panels` directory
- A template for the required custom instrument panel callback sequence file named CustomInstrumentPanel.seq located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory
- A template for the required corresponding custom instrument panel LabVIEW project files located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates\CustomInstrumentPanelTemplate` directory

Note To modify the installed examples or templates, copy the files from the existing locations to the `<TestStand Public>` directory and make changes to the copies of the files.

Toolbar Improvements

You can now use the TSM toolbar buttons instead of the default TestStand toolbar buttons to resume sequence execution from a breakpoint for lot or part testing.

LabVIEW VI Analyzer Test

TSM now installs the following test, enabled by default, to the LabVIEW VI Analyzer in the **TestStand Semiconductor Module** section:

- **For Loop Error Handling**—Verifies error handling in VIs that use For Loops. Confirms that errors from the For Loop are merged with any errors that occurred before the loop executed to ensure that errors that occur before the loop executes are propagated correctly.

Refer to the **LabVIEW Help** for more information about the LabVIEW VI Analyzer. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

Part Average Testing Support

Part average testing (PAT) is a method based on statistical analysis to identify and fail parts that have characteristics significantly outside the normal distribution of other parts in the same lot. TSM does not install a default implementation of part average testing. You must use the TSM PAT plug-in architecture to customize and perform part average testing with TSM. The TSM PAT plug-ins include a required PAT callback sequence file and corresponding code modules. The PAT callback sequence file contains PAT entry point sequences that TSM calls during execution to accomplish part average testing. Use the example PAT plug-in, located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In` directory, as a starting point for custom PAT plug-ins you create.

Refer to the Part Average Testing Examples for information about enabling and performing part average testing (PAT) in a test program.

Additional Enhancements

Additional Improvements

The NI TestStand 2016 SP1 Semiconductor Module includes the following additional enhancements:

- The Standard Test Data Format (STDF) Log result processing plug-in now generates summary records for the Hardware Bin Record (HBR), Software Bin Record (SBR), Test Synopsis Record (TSR), and Part Count Record (PCR) records. The records are included at the end of the STDF log file and have a HEAD_NUM value of 255 to indicate that they are summary records. Visit `ni.com/info` and enter the Info Code `exr3v6` for information about disabling these summary records.
- TSM now uses the `EXEC_TYPE` and `EXEC_VER` fields in the Master Information Record (MIR) of the STDF log file to record whether STS Software is installed and which version is installed.
- You can now log results only when a DUT fails. Use the **Log Results Only for DUT Failures** option in the Test Results Logs Options dialog box during debugging to record only failures instead of all test results.
- You can now use the `AvailableSiteNumbers` property on the NI_SemiconductorModule_StationSettings data type to specify which site numbers from a pin map for a test program to use when running the test program. For example, you can set this property to disable specific sites or to use the particular connections of a pin map that match the DIB for the test station. The default TSM operator interfaces and the Lot Statistics Viewer display only the sites you specify. Update existing custom LabVIEW or C# operator interfaces to display the configured site numbers when you use the AvailableSites station setting. Additionally, you can modify the ConfigureStationSettings callback to provide a custom operator interface to set the AvailableSites station setting.

  Ensure that any custom handler driver you created reads the Boolean values from the **RequestedSiteState** parameter in the StartOfTest callback to determine which sites to test. If you use the `AvailableSiteNumbers` property to specify the set of site numbers for the test program to use, the **RequestedSiteState** array contains `True` values for each site number the test uses and `False` values for each site number the test does not use.

- In the Pin Map Editor, you can now manually enter relative file paths or manually modify existing file paths to be relative. Browsing to a file using the Select Pin Map File dialog box always uses an absolute path.

The Get All FPGA Instrument Names, Get All NI-RFPM Instrument Names, Get All NI-RFPM De-embedding Data, Get All NI-5530 RF Port Module Names, and Pin(s) To NI-RFPM Sessions TSM Code Module API VIs and the GetFpgaInstrumentNames, GetNIRfpmInstrumentNames, GetNIRfpmSessions, and GetAllNIRfpmDeembeddingData TSM Code Module API .NET class library methods now resolve file paths differently. If the path is an absolute path, the VI or method returns the absolute path whether the file exists or not. If the path is a relative path and the file exists relative to the path of the pin map file, the VI or method returns the absolute path of the existing file. Otherwise, the VI or method returns a string without error. This change in behavior breaks any test programs that use the previous versions of these VIs or methods that expect relative file paths to remain unresolved.

- You can now specify and obtain the orientation of data in S2P files that characterize the de-embedding network for each port.

The pin map XML schema now includes a `deembeddingOrientation` attribute on the `Connection`, `SystemConnection`, and `MultiplexedDUTPinRoute` elements that you can use with the `deembeddingFilePath` attribute to specify the orientation of the data in the S2P file relative to the port the `channel` attribute specifies. Valid values are **Port1TowardDUT** or **Port2TowardDUT**.

The Get All NI-RFPM De-embedding Data VI and the Pin(s) To NI-RFPM Sessions VI now include a **De-embedding Files** indicator that returns the path and orientation of the data in the S2P file relative to the port you specify. The connector pane for these VIs have changed, which will not break existing code. However, if you want to use the new **De-embedding Files** indicator, you must replace existing VIs with the new versions of the VIs on the NI-RFPM VIs palette.

- You can now use the **Select All** checkboxes on the DUT Pins, System Pins, or Pin Groups tabs of the Pin Map tab of the Pin Map Editor to add or remove pins from the pin group.
- Refer to the Query Pin/Site Measurement for Unsupported Measurement Type example in the Multisite Programming Scenarios example for information about how to query for measurement data for a pin and site combination if the Extract Pin Data VI does not support the measurement type.

See Also

NI TestStand 2016 SP1 Semiconductor Module Compatibility Issues

NI TestStand 2016 SP1 Semiconductor Module Compatibility and Known Issues

Refer to the List of Known Issues in NI TestStand 2016 SP1 Semiconductor Module on the NI website for a list of known issues in NI TestStand 2016 SP1 Semiconductor Module (TSM).

The NI TestStand 2016 SP1 Semiconductor Module introduces the following behavior changes between version 2016 and version 2016 SP1:

- The default TSM operator interfaces display only the sites you specify when you use the `AvailableSiteNumbers` property on the NI_SemiconductorModule_StationSettings data type to specify which site numbers from a pin map for a test program to use when running the test program. The TSM 2016 and earlier default operator interfaces and custom operator interfaces based on those versions display site numbers starting at 0 and increasing by 1, up to the number of sites. You must update existing custom LabVIEW or C# operator interfaces to display the configured site numbers when you use the AvailableSites station setting.
- In previous versions of TSM, the **RequestedSiteState** array in the StartOfTest callback contained only `True` values. Ensure that any custom handler driver you created reads the Boolean values from the **RequestedSiteState** parameter in the StartOfTest callback to determine which sites to test. If you use the `AvailableSiteNumbers` property to specify the set of site numbers for the test program to use, the

RequestedSiteState array contains `True` values for each site number the test uses and `False` values for each site number the test does not use.

- The version number of the pin map XML schema changed from `1.2` to `1.3`. The schema is not backward compatible with NI TestStand 2016 Semiconductor Module.
- The Get All FPGA Instrument Names, Get All NI-RFPM Instrument Names, Get All NI-RFPM De-embedding Data, Get All NI-5530 RF Port Module Names, and Pin(s) To NI-RFPM Sessions TSM Code Module API VIs and the GetFpgaInstrumentNames, GetNIRfpmInstrumentNames, GetNIRfpmSessions, and GetAllNIRfpmDeembeddingData TSM Code Module API .NET class library methods now resolve file paths differently. If the path is an absolute path, the VI or method returns the absolute path whether the file exists or not. If the path is a relative path and the file exists relative to the path of the pin map file, the VI or method returns the absolute path of the existing file. Otherwise, the VI or method returns a string without error. This change in behavior breaks any test programs that use the previous versions of these VIs or methods that expect relative file paths to remain unresolved.
- The Get All NI-RFPM De-embedding Data VI and the Pin(s) To NI-RFPM Sessions VI now include a **De-embedding Files** indicator that returns the path and orientation of the data in the S2P file relative to the port you specify. The connector pane for these VIs have changed, which will not break existing code. However, if you want to use the new **De-embedding Files** indicator, you must replace existing VIs with the new versions of the VIs on the NI-RFPM VIs palette.
- TSM now uses the `EXEC_TYPE` and `EXEC_VER` fields in the Master Information Record (MIR) of the STDF log file to record whether STS Software is installed and which version is installed.

NI TestStand 2016 SP1 Semiconductor Module Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2016 SP1 Semiconductor Module on the NI website for a list of bugs fixed in NI TestStand 2016 SP1 Semiconductor Module.

This list is not exhaustive. If you reported an issue to NI and received a CAR ID to track the issue, you can search for the CAR ID in the ID or Legacy ID column to determine whether the issue has been fixed.

What's New in the NI TestStand 2016 Semiconductor Module

The NI TestStand 2016 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2016 Semiconductor Module.

NI TestStand 2016 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2016 Semiconductor Module (TSM) and other changes since the NI TestStand 2014 Semiconductor Module SP1.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Improved Test Program Development

Semiconductor Action Step

Use the Semiconductor Action step to perform an action, such as instrument configuration, with access to the pin map and per-site inputs. You can configure multisite and per-site input options directly on the step.

Support for the Digital Pattern Editor and the NI-Digital Pattern Driver

- Select **Semiconductor Module»Launch Digital Pattern Editor** or click the **Launch Digital Pattern Editor** button on the TSM toolbar to open digital pattern project files in the Digital Pattern Editor.
- Use the Digital Pattern Project panel of the Test Program Editor to specify the pathname of the digital pattern project file to use in the test program. You can use the Specifications Files panel of the Test Program Editor to view the specifications files in a digital pattern project.

- The Pin Map schema now includes native support for Digital Pattern instruments with the `NIDigitalPatternInstrument` element.
- The NI TestStand 2016 Semiconductor Module natively supports digital pattern instruments that use the NI-Digital Pattern Driver, such as the PXIe-6570, and legacy digital waveform instruments that use the NI-HSDIO driver, such as the PXIe-6556. Use the TSM Code Module API that corresponds to the type of digital instrument the test system includes. For example, use the NI-Digital Pattern VIs to manage digital pattern instruments and sessions, to manage digital pattern waveform data, and to access digital pattern project files.

Note The VIs on this palette are available only in 64-bit LabVIEW.

- The Accelerometer and Multisite Programming Scenarios examples now use the NI-Digital Pattern driver. Versions of these examples that use the NI-HSDIO driver remain available.

Customizing Operator Interface Run-Time Error Behavior

The default operator interfaces now write a message to an error log file and continue testing when the `MainSequence` sequence returns a run-time error from a source other than TSM, such as a code module or an instrument driver. You can change the operator interface run-time error behavior by using the two new TSM Application API properties EndLotOnCodeModuleRuntimeError and DisplayDialogOnCodeModuleRuntimeError to specify whether to end the lot and display error dialog boxes when the `MainSequence` sequence returns a run-time error from a source other than TSM.

C# NI-RFmx and NI-RFPM TSM Code Module API Support

The TSM Code Module API now includes C# implementations for NI-RFmx and NI-RFPM.

Additional Enhancements

Additional Improvements

The NI TestStand 2016 Semiconductor Module includes the following additional enhancements:

- **Expression to Determine Test Number at Run Time**—The `Result.Evaluations` property of the Semiconductor Multi Test step type includes a new `TestNumberExpr` field, which is an expression that determines the test number at run time. If this expression is not empty, the Semiconductor Multi Test step evaluates the expression and copies the evaluated value to the `TestNumber` property.
- **Indicating End of Wafer using Batch Process Model**—Prober drivers no longer need to set the **EndOfWafer** parameter of the EndOfTest handler/prober driver entry point to indicate the end of each wafer when using the Batch process model. The NI TestStand 2016 Semiconductor Module determines the end of the wafer from the **WaferRuntimeData.StartOfWafer** parameter in the StartOfTest handler/prober driver entry point. When using the Batch process model, omitting the wait for an end-of-wafer status message to set the **EndOfWafer** parameter in the EndOfTest entry point might improve performance of the test system.

  Prober drivers must set the **EndOfWafer** parameter at the end of each wafer when using the Sequential process model.
- **Bin Types Passed to Handler**—TSM passes an array of bin types to the EndOfTest handler/prober driver entry point. Each element in the array indicates the type of bin (Pass, Fail, and so on) of the corresponding element in the **SoftwareBinData** and **HardwareBinData** parameter arrays.
- **Temporary STDF File**—During testing, the STDF Log result processor writes data to a temporary file with an extension of `.stdtemp` in the destination directory you specify in the STDF Log Options dialog box. When the file completes, the STDF Log result processor renames the file to the final report filename you specify.

See Also

NI TestStand 2016 Semiconductor Module Compatibility Issues

NI TestStand 2016 Semiconductor Module Compatibility and Known Issues

Refer to the List of Known Issues in TestStand NI 2016 Semiconductor Module on the NI website for a list of known issues in NI TestStand 2016 Semiconductor Module (TSM).

The NI TestStand 2016 Semiconductor Module introduces the following behavior changes between version 2014 SP1 and version 2016:

- The default TSM operator interfaces no longer display an error message and end testing when the `MainSequence` sequence returns a run-time error from a source other than TSM, such as a code module or an instrument driver. Instead, the default operator interfaces now write the error message to an error log file and continue testing the lot. The change in behavior does not affect operator interfaces based on source code from previous versions of TSM. You can change the operator interface run-time error behavior by setting TSM Application API properties on or by handling events from the SemiconductorModuleManager object in the operator interface source code. By default, when any run-time error occurs, TSM assigns the current part to the Default Error bin the bin definitions file specifies.
- The file extensions for pin map files and bin definitions files have changed from `.xml` to `.pinmap` and `.bins`, respectively. You can continue to load pin map files and bin definitions files with `.xml` file extensions without error or warning.
- The Accelerometer examples, located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer` directory, now use the NI-Digital Pattern driver. Refer to the examples in the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer - Legacy Digital` directory for Accelerometer examples that use the NI-HSDIO driver.
- TSM detects a retested part by comparing the part IDs or die coordinates that the handler or prober provides for the part to the part IDs and die coordinates that the handler or prober provided for previously tested parts. TSM determines the number of passing and failing parts and the number of parts per bin based on the test results of the last time it tested the part.

In the STDF log file, TSM sets the PART_FLG field of the Part Result Records (PRR) that represent results of retested parts and sets the RTST_CNT field of the Wafer Result Record (WRR) and the Part Count Record (PCR) to the number of parts retested one or more times.

Previous versions of TSM did not distinguish between the first test of a part and subsequent tests of the same part except when you clicked the **Retest** toolbar button in the sequence editor. As a result, the part counts that previous versions of TSM generate differ from the part counts that the NI TestStand 2016 Semiconductor Module generates if the handler or prober retests one or more parts. The affected STDF fields include the following:

- PRR.PART_FLG
- WCR.GOOD_CNT
- WCR.PART_CNT
- WCR.RTST_CNT
- PCR.GOOD_CNT
- PCR.PART_CNT
- PCR.RTST_CNT
- HBR.HBIN_CNT
- SBR.SBIN_CNT

- The name of the NamespacedSymbol(s)ToValue(s) VI on the Specifications palette has changed to Get Specification(s) Value(s). The name change does not require changes to VIs that use the old name.

NI TestStand 2016 Semiconductor Module Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2016 Semiconductor Module on the NI website for a list of bugs fixed in NI TestStand 2016 Semiconductor Module.

This list is not exhaustive. If you reported an issue to NI and received a CAR ID to track the issue, you can search for the CAR ID in the ID or Legacy ID column to determine whether the issue has been fixed.

What's New in the NI TestStand 2014 Semiconductor Module SP1

The NI TestStand 2014 Semiconductor Module SP1 introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2014 Semiconductor Module SP1.

NI TestStand 2014 Semiconductor Module SP1 New Features

The following list describes the new features in the NI TestStand 2014 Semiconductor Module SP1 (TSM) and other changes since the NI TestStand 2014 Semiconductor Module.

 Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Improved Test Program Development

▾ RFmx Pin Map API

The Pin Map API now supports the RFmx Instrument Sessions.

▾ NI-RFPM Pin Map API

The Pin Map API now supports the RF Port Module (NI-RFPM) API. This API serves as a replacement for the NI-5530 Port Module API. It provides several usability enhancements for developing multi-port RF measurements.

▾ Custom Model Plug-in API

TSM now has an easier way to access lot statistics, start times, end times, wafer information, and handler/prober information in a custom result processor so you can create custom reports and log files.

Performance Improvements

▾ Execution Profiler

Use the Execution Profiler window to view and record duration of steps, code modules, and other resources a multithreaded TestStand system uses over a period of time. You can review the recorded data in graphs and sortable tables to identify

performance bottlenecks and design flaws and to gain insight into the behavior and timing of complex multithreaded systems. You can copy the information to external applications, such as Microsoft Word or Excel.

Additional Enhancements

Additional Improvements

NI TestStand 2014 Semiconductor Module SP1 includes the following additional enhancements:

- **Base Deployment License**—The NI TestStand Semiconductor Module Base Deployment License is the minimum license required for all deployed TSM-based applications. Activate this license to deploy the NI TestStand Semiconductor Module Runtime, the TestStand Runtime, and the Switch Executive Runtime. The Base Deployment License enables you to run a TSM operator interface and test programs on the single test station to which the license applies. This license does not grant the ability to perform any development tasks using the TestStand Sequence Editor, a TestStand custom sequence editor, or the TestStand API.
- **Improved Getting Started Content**—The getting started content now includes a brief tour of TSM, an overview of test program components, and a new example that demonstrates several features of TSM in a test program that makes common measurements to test an imagined accelerometer part.
- **Start Lot Without Configuring**—You can customize the operator interface to run without requiring a **Configure Lot** button by implementing the ConfigureLotSettings callback to programmatically set the LotSettings instead of using a dialog box.
- **End of Test dialog box**—This dialog box launches at the end of each test and displays hardware and software bin information on a per test or per lot basis when you use the Batch or Sequential process model with the NI Simulated Handler Driver.
- **Toolbar Enhancements**—You can now use the **Selected Configuration** control on the TSM toolbar to select the test program configuration to use when testing. The available items correspond to the configurations in the

active sequence file. The value initially corresponds to the value of the `LotSettings.Standard.ActiveConfigurationName` property in the lot settings. If the active sequence file does not contain a configuration that corresponds to the `ActiveConfigurationName` lot setting, the control displays one of the configurations in the sequence file. Changing the selected configuration with this control does not modify the `ActiveConfigurationName` lot setting. You can also use the Configure Lot Settings dialog box to change the test program configuration.

- **.NET (C#) Example Code**—Code module development topics include .NET example code.

See Also

NI TestStand 2014 Semiconductor Module SP1 Compatibility Issues

NI TestStand 2014 Semiconductor Module SP1 Compatibility and Known Issues

Refer to the List of Known Issues in NI TestStand 2014 Semiconductor Module SP1 on the NI website for a list of known issues in NI TestStand 2014 Semiconductor SP1 (TSM).

The NI TestStand 2014 Semiconductor Module SP1 introduces the following behavior changes between version 2014 and version 2014 SP1:

- The method for creating custom DTRs has changed when you enable the **Generate One File per Wafer** option.
- The EndOfTest handler/prober driver entry point includes a new EndOfWafer parameter, which must be set to `True` at the end of each wafer for the **Generate One File per Wafer** option to function correctly and to generate Wafer Result Records (WRR).
- To better comply with STDF standards, the START_T field of the Master Information Record (MIR) is recorded at the time the handler or prober sends the initial start-of-test (SOT) signal. In NI TestStand 2014 Semiconductor Module, the START_T field contains the time that the operator clicked the Start Test button.

- The `SemiconductorModuleContext` object on any instance of a Semiconductor Multi Test step includes only the pins specified on the Options tab unless you call the step from a process model callback sequence, such as `ProcessSetup` and `ProcessCleanup`, in which case the `SemiconductorModuleContext` object includes all pins in the pin map.

  In previous versions of TSM, when executing with the Sequential or Parallel process models, `SemiconductorModuleContext` objects included all the pins in the pin map regardless of the settings on the Options tab. Semiconductor Multi Test steps that execute without errors using the Sequential or Parallel process models in NI TestStand 2014 Semiconductor Module might generate run-time errors in NI TestStand 2014 Semiconductor Module SP1 if the code module for the step attempts to use pins that you did not specify on the Options tab. To correct the error, specify all necessary pins on the Options tab of the Semiconductor Multi Test step.

NI TestStand 2014 Semiconductor Module SP1 Bug Fixes

Refer to the List of Bugs Fixed in NI TestStand 2014 Semiconductor Module SP1 on the NI website for a list of bugs fixed in NI TestStand 2014 Semiconductor Module SP1.

This list is not exhaustive. If you reported an issue to NI and received a CAR ID to track the issue, you can search for the CAR ID in the ID or Legacy ID column to determine whether the issue has been fixed.

What's New in the NI TestStand 2014 Semiconductor Module

The NI TestStand 2014 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2014 Semiconductor Module.

NI TestStand 2014 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2014 Semiconductor Module (TSM) and other changes since the NI TestStand 2013 Semiconductor Module.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Improved Test Program Development

Pin Map Editor

Use the Pin Map Editor to view, create, modify, and save pin map files instead of editing the XML files directly. Use the Pin Map panel in the Edit Test Program dialog box to specify a pin map file for a test program.

Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to launch the Pin Map Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Pin Map** in the Edit Test Program dialog box to launch the Pin Map panel. Click the **Open file for edit** button to launch the Pin Map Editor.

Bin Definitions Editor

Use the Bin Definitions Editor to view, create, modify, and save bin definitions files instead of editing the XML files directly. Use the Bin Definitions panel in the Edit Test Program dialog box to specify a bin definitions file for a test program.

Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar to launch the Bin Definitions Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Bin Definitions** in the Edit Test Program dialog box to launch the Bin Definitions panel. Click the **Open file for edit** button to launch the Bin Definitions Editor.

Pin Groups

You can now create and use pin groups in the pin map file, the Pin Map Editor, the Semiconductor Multi Test step, and code modules.

Per-Site and Instrument Session Data Support

Use the following new TSM Code Module API VIs or .NET methods to associate or return data for a site or instrument session:

- **Set Site Data**—Associates a data item with each site. You can associate data with all sites or with the subset of sites in the **Semiconductor Module context**. You can use this VI or .NET method to store instrument sessions or other per-site data you initialize in a central location but access within each site.
- **Get Site Data**—Returns per-site data that a previous call to the Set Site Data VI or .NET method stores. The returned array contains the data the **Semiconductor Module context** stores for each site in the same order as the sites the Get Site Numbers VI or .NET method returns.
- **Set Global Data**—Associates a data item with a data ID. You can use this VI or .NET method to store an instrument session or other data you initialize in a central location but access from multiple sites. The data item is accessible from a process model controller execution and all of its test socket executions.
- **Get Global Data**—Returns a global data item that a previous call to the Set Global Data VI or .NET method stores.

External Specifications

Use a specifications file to define a set of symbols and associated numeric values that you can reference in test programs and code modules to set limits and testing specifications. You can specify the values directly in the specifications file or calculate the values from other symbols in the file using formulas you write with simple arithmetic functions.

Use the NamespacedSymbol(s)ToValue(s) TSM Code Module API VI or the GetSpecificationsValue or GetSpecificationsValues TSM Code Module API .NET methods to query one symbol or an array of symbols and return the calculated value or values.

The specifications XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\Specifications.xsd`, defines the structure for a specifications XML file. Use the **Specifications File Path** control on the Specifications Files panel of the Edit Test Program dialog box to specify one or more specifications files to use with the test program.

Multisite Programming Scenarios

The new Multisite Programming Scenarios example, located in <TestStand Public>\Examples\NI_SemiconductorModule\Multisite Programming Scenarios\MultisiteScenarios.seq demonstrates how to address several multisite use cases.

Protecting Test Limits

To protect test limits files from modification or viewing, use the **Embed Test Limits File** option on the Test limits Files panel of the Edit Test Program dialog box to embed the external test limits files in the test program sequence file before you password-protect the test program sequence file. Use the **Extract Test Limits File** option on the Test limits Files panel to extract an embedded test limits file from the test program sequence file to view or change the contents of the test limits file.

Note NI does not recommend using passwords as the only way of protecting intellectual property.

Dragging and Dropping Variables

You can now drag and drop TestStand variables into the expression controls on the Tests tab and the Per-Site Inputs tab of the Semiconductor Multi Test step and the Get Test Settings tab of the Get Test Settings step.

Improved Debugging

Debugging Sequences

Use the TSM toolbar buttons instead of the default TestStand toolbar buttons to control execution and view lot statistics while debugging a sequence. The TSM toolbar includes new Lot Execution Control buttons for starting a single test, starting or resuming a lot, pausing a lot, retesting a single DUT, ending a lot, and launching a Lot Statistics Viewer.

You can also select **Semiconductor Module»Show Lot Statistics Viewer** to launch the Lot Statistics Viewer, in which you can control execution and view lot statistics while debugging a sequence. The Lot Statistics Viewer displays a new tab for each test program sequence file you execute. The tab includes a table of the

software bin statistics for each site and highlights the cell for each updated DUT result. When execution completes, the table dims.

New Run-Time Error Option

The Semiconductor Module Run-Time Error dialog box contains a new run-time error handling option. Select the **End lot after running cleanup** option to end lot testing after the execution proceeds to the Cleanup step group for the sequence.

Improved Wafer Testing Support

Additional STDF Wafer Records

You can now update fields in the Wafer Configuration Record (WCR), Wafer Information Record (WIR), and Wafer Result Record (WRR) of an STDF log file. The Setup and the StartOfTest handler/prober driver entry points include wafer-related parameters and the NI_SemiconductorModule_LotSettings data type includes wafer-related properties to support wafer testing.

Operator Interface Improvements

Operator Interface Enhancements

The NI TestStand 2014 Semiconductor Module operator interface includes the following enhancements

- **Easier customization**—The redesigned default operator interface uses an enhanced TSM Application API to simplify the customization process. You can continue to use NI TestStand 2013 Semiconductor Module operator interfaces in NI TestStand 2014 Semiconductor Module.
- **Settings table improvements**—The settings table on the right side of the default operator interface displays common lot and station settings. You can configure the list of settings to display in the table by editing the `OISettingsTable.cfg` file located in the `<TestStand Application Data>\Cfg\NI_SemiconductorModule` directory. The `<TestStand>\Components\Schemas\NI_SemiconductorModule\OISettingsTable.xsd` schema file describes the format of the configuration file.

- **Mid-Lot Summary report**—You can now refresh the Mid-Lot Summary text report during execution.

C# Support

C# Operator Interface

The NI TestStand 2014 Semiconductor Module includes a C# implementation of the default operator interface. The source code is located in the `<TestStand>\User Interfaces\NI_SemiconductorModule\CSharp` directory.

C# API

The TSM Code Module API and TSM Application API include C# implementations.

C# Examples

The Multisite Simple Flow example now includes a C# implementation.

Other Enhancements

64-Bit TestStand Support

TSM now supports 64-bit TestStand. Refer to the 64-bit TestStand and Migrating from 32-bit TestStand book of the **TestStand Help** for more information about the differences between 32-bit TestStand and 64-bit TestStand.

Note You must use 32-bit TestStand and 32-bit LabVIEW with the Switch Executive.

Multiple Versions of LabVIEW

TSM now supports LabVIEW 2014 SP1 in addition to supporting LabVIEW 2013 and LabVIEW 2013 SP1.

Additional STDF Support

TSM includes support for the following additional STDF fields:

- **Part Results Record**—The STDF Log result processing plug-in sets the PART_ID, PART_TXT, X_COORD, and Y_COORD fields in the Part Results Record of the STDF version 4 specification by using the values of properties on the UUT data type. The handler/prober driver StartOfTest entry point can set any

of these fields by setting the values of the output parameters to the StartOfTest entry point.

- **Additional NI_SemiconductorModule_LotSettings Data Type Properties**—The NI_SemiconductorModule_LotSettings data type includes new properties to support Master Information Record (MIR) and Wafer Configuration Record (WCR) fields.
- **Datalog Text Records (DTR)**—You can insert DTR in various locations in the STDF log file. The method for creating DTRs differs depending on the locations at which you insert the DTR.

You can now use the **Limit Number of Test Data Records** option in the STDF Log Options dialog box to limit the number of individual part test records in the STDF log to one out of every **N** DUTs you specify per site. Individual test records include PTR, FTR, and DTR. The summary test result records include the test result records of the parts for which you omitted individual test records.

Additional Handler/Prober Support

TSM now passes software bin numbers to the handler/prober driver in addition to hardware bin numbers.

VI and Palettes

The **TestStand Semiconductor Module»Pin Map** Functions palette in LabVIEW has been renamed to the **TestStand Semiconductor Module»Code Module Development** Functions palette.

Use the following new VI on the **TestStand Semiconductor Module»Code Module Development»Specifications** subpalette to work with specifications files:

- **NamespacedSymbol(s)ToValue(s)**—Returns the value or values calculated for the namespaced symbol or symbols in the Semiconductor Module context specifications file. Adapts automatically to a single symbol or an array of symbols.

Missing Tests or Steps Return Error when Importing Limits File

Enable the **Require every step to be in test limits file** option in the Import Test Limits into Sequence File dialog box or the Configuration panel of the Edit Test Program dialog box for the Import/Export Test Limits tool to return an error and not import the file if a test or step exists in the sequence but does not exist in the limits file. If you do not enable this option, the Import/Export Test Limits tool imports only matching tests or steps in the limits file.

▾ Launching Examples from the Help

You can now launch an example directly from the corresponding help topic by clicking the **Open Example** button in the help topic.

See Also

NI TestStand 2014 Semiconductor Module Compatibility Issues

NI TestStand 2014 Semiconductor Module Compatibility Issues

The NI TestStand 2014 Semiconductor Module (TSM) introduces the following behavior changes between version 2013 and version 2014:

- The version number of the pin map XML schema and the bin definitions XML schema both changed from `1.0` to `1.1`. The schemas are not backward compatible with NI TestStand 2013 Semiconductor Module.
- Pin and pin group names must begin with a letter or underscore (_) and are limited to A-Z, a-z, 0-9, or _ characters.
- The following filenames and directory locations changed to more accurately reflect that the TSM handler/prober driver communicates with handlers and probers:

Previous Filename/Location	Current Filename/Location
<TestStand>\Components\Modules\NI_SemiconductorModule\Handlers	<TestStand>\Components\Modules\NI_SemiconductorModule\HandlersAndProbers
<TestStand>\Components\Modules\NI_SemiconductorModule\Templates\HandlerDriver.seq	<TestStand>\Components\Modules\NI_SemiconductorModule\Templates\HandlerProberDriver.seq

<TestStand Public>\Components\Modules\NI_SemiconductorModule\Handlers	<TestStand Public>\Components\Modules\NI_SemiconductorModule\HandlersAndProbers

- You no longer use the `SemiOIOptions.ini` file to specify settings to display in the operator interface. Use the `OISettingsTable.cfg` file instead.
- When you use reentrant VIs and subVIs, NI recommends using only the **Shared clone reentrant execution** reentrancy execution option. Selecting the **Preallocated clone reentrant execution** option might negatively affect performance.
- When you pause a lot, TSM 2013 or earlier pauses execution of the lot after sending the bin results to the handler or prober with the end-of-test (EOT) signal. In TSM 2014 or later, pausing a lot pauses execution after testing of the current batch of DUTs is complete but before sending the bin results to the handler or prober with the EOT signal. Operator interfaces built using TSM 2013 or earlier pause execution after sending the bin results to the handler or prober and do not include the ability to retest a DUT.

What's New in the NI TestStand 2013 Semiconductor Module

The NI TestStand 2013 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2013 Semiconductor Module.

NI TestStand 2013 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2013 Semiconductor Module (TSM) and other changes since the NI TestStand 2012 R2 Semiconductor Module.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

▾ Test Program Enhancements

In addition to specifying pin map and bin definitions files, the test program now includes test program configurations, which you use to specify test conditions and an external test limits file for the test program to load when executed.

You can now use the Edit Test Program dialog box to specify the pin map and bin definitions files, create and edit test program configurations, and configure other settings for the test program.

Select **Semiconductor Module»Edit Test Program: <filename>** or click the **Edit Test Program: <filename>**  button on the TSM toolbar to launch the Edit Test Program dialog box for the sequence file.

Exporting and Importing Test Limits with Text Files

You can use a tab-delimited text file to export and import test limits from and to a Semiconductor Multi Test step in a single sequence file at edit time or run time. For example, during development, you can export all the tests in a test program to review and edit and then import the changes back into the test program. At run time, you can load and execute a different set of test limits from separate text files based on the test program configuration you select by exporting the test limits and creating multiple copies of the file to edit for each unique set of test limits you want to use.

Select **Semiconductor Module»Export Test Limits from <filename>** or **Import Test Limits into <filename>** or click the **Export Test Limits from <filename>**  button or the **Import Test Limits into <filename>**  button on the TSM toolbar to export test limits from a sequence file into a tab-delimited test limits text file or to import test limits from a tab-delimited test limits text file into a sequence file. When you import test limits from a text file, you can update limits in matching tests or replace all tests in matching steps.

Refer to the Importing Test Limits from a File tutorial for more information about importing test limits from and to a Semiconductor Multi Test step.

Scaling Measurement and Limit Data

You can specify a scaling factor to select the units you want to use to display and specify limit values and to display measurement values. Use the Scaling Factor column in the Tests table on the Tests tab of the Semiconductor Multi Test step to

specify the scaling factor for the limits and measurements for each test. TSM assumes that all measurement values that code modules publish use base units.

By default, the limits and the measurement use the same scaling factor. Changing the scaling factor in the Scaling Factor column in the Tests table sets the scaling factor for the limits and for the measurement. Use the Property Browser panel on the Properties tab of the Step Settings pane to set the appropriate properties to specify different scaling factors for each limit and the measurement.

Semiconductor Multi Test Step Edit Tabs Enhancements

The Semiconductor Multi Test step edit tabs include the following enhancements:

- Use the Per-Site Inputs tab to configure the per-site inputs for the step. Each per-site input specifies a data value that you access in a code module using the TSM pin map VIs.
- You can copy and paste data in the Tests table on the Tests tab in the following ways:
  - Press <Ctrl-C> to copy the contents of selected table rows, columns, or cells to the clipboard.
  - Press <Ctrl-V> to insert data from the clipboard into the table starting from the top left selected cell. If the number of rows of data on the clipboard is greater than the number of tests in the Tests table, the paste command adds new tests to the step to match the clipboard data.
  - You can copy data to tests on the same Semiconductor Multi Test step, to tests on other Semiconductor Multi Test steps, or to a Microsoft Excel spreadsheet.
  - When you modify the data in Excel and paste it back to a Semiconductor Multi Test step, TSM returns an error if the modified data is invalid for a test.
- The Options tab includes the following enhancements:
  - Use the **Test Numeric Display Format** control to specify the TestStand numeric format associated with the limits and measurement data. The numeric format determines how the Tests tab displays numeric values and how the TestStand report generators display numeric values in reports. Click

the **Edit** button to launch the Numeric Format dialog box, in which you can specify the numeric format.

- When you associate a pin map file with the sequence file, the **Multisite Execution Diagram** section shows the threads the step uses to execute code modules when executing with the Batch process model using multiple sites. Each rectangle represents a single thread. The numbers within the rectangle represent the sites tested in the same thread. Each thread corresponds to one of the test socket threads. The step determines which test socket thread executes the code module at run time.
- The **SemiconductorModuleContext Pins** option shows the list of pins the `SemiconductorModuleContext` object contains at run time. By default, the `SemiconductorModuleContext` contains all pins in the pin map file. You can improve performance in some situations by specifying only the pins you access in the code module. Use the **Include System Pins** and **Specify DUT Pins** options to specify individual pins in the pin map file to include in the `SemiconductorModuleContext` object.

Get Test Settings Step

Use the Get Test Settings step to obtain the values for lot settings, station settings, or custom test settings. Store the values of the settings in TestStand local variables so that any step in the sequence can access the setting values.

Use the Get Test Settings edit tab in the TestStand Sequence Editor to specify the list of lot settings, station settings, and custom test settings and the locations to store the values.

Pin Map VIs and Palettes

TSM adds the following new subpalettes:

- **RF subpalette**—Use the NI-RFSA, NI-RFSG, FPGA, and NI-5530 RF Port Module VIs to manage NI-RFSA, NI-RFSG, FPGA, and NI-5530 RF Port Module instruments and sessions.

TSM VIs include the following changes:

- **Get Pin Names VI and Filter Pins VI**—Use the **Capability** control to limit the filtered pins to those connected to a channel that defines the capability you specify. Use **Capability** to differentiate between pins in the same instrument with different capabilities, such as NI-HSDIO Dynamic DIO channels and PFI lines. If a pin is connected to channels in which the capability is define only for a subset of sites, the VI returns an error. Pass an empty string to return all elements in **Pins** that match **Instrument Type Id**.
- **Publish Data VIs**—Redesigned Publish Data VI polymorphic instances use a Pin Query Context object that tracks the sessions and channels associated with a pin query. TSM uses this object to publish measurements, extract data from a set of measurements, and create or rearrange waveforms.

TSM adds the following new VIs:

Use the Advanced:Per-Site:Boolean, Advanced:Per-Site:Double, and Advanced:Per-Site:String polymorphic instances of the Publish Data VI to publish Boolean measurement data, double-precision, floating-point measurement data, or strings for all sites in the **Semiconductor Module context**.

Use the Get Input Data VI to return per-site input data as defined in the Semiconductor Multi Test step.

Deprecated VIs

The following VIs are now deprecated. NI supports these VIs but recommends that you update files to reflect these changes to ensure compatibility with future versions of TSM.

Deprecated VI	Preferred VI
Create Multisite Digital Waveforms (Deprecated)	Create Multisite Digital Waveforms
Get Session And Channel Index (Deprecated)	Get Session And Channel Index
Pins to NI-HSDIO Channel Masks (Deprecated)	Get NI-HSDIO Channel Masks
Publish Data (Deprecated)	Publish Data
Rearrange Multisite Digital Waveforms (Deprecated)	Rearrange Multisite Digital Waveforms

The preferred replacement VIs use a Pin Query Context object that tracks the sessions and channels associated with a pin query. TSM uses this object to publish

measurements, extract data from a set of measurements, and create or rearrange waveforms.

Pin Map File XML Structure

The pin map file XML structure includes the following new elements and attributes:

New Element or Attribute	Description
`PFILines` attribute of existing `<NIHSDIOInstrument>` element	(Optional) Defines the PFI lines available in the NI-HSDIO instrument in a comma-separated list of numbers or ranges of numbers separated by a hyphen. PFI number ranges are inclusive and must be in ascending order. Example: `PFILines=2,3,4-8`
`<NIRFSAInstrument>` element and `name` attribute	Defines an NI-RFSA instrument.
`<NIRFSGInstrument>` element and `name` attribute	Defines an NI-RFSG instrument.
`<NIVSTInstrument>` element and `name` and `fpgaFilePath` attributes	Defines an NI-VST instrument that can hold RFSA, RFSG, and FPGA sessions.
`<NI5530RFPortModule>` element and `name` and `calibrationFilePath` attributes	Defines an NI-5530 RF Port Module instrument. You can use the NI-5530 RF Port Module to multiplex one RF instrument across multiple test sites or multiple RF instruments across multiple test sites.
`multiplexerTypeId` attribute of existing `<Multiplexer>` element	String that identifies the switch type, family, class, or product group. You cannot specify a value that begins with `ni`. This value is a string that you define in the pin map and is not a predefined value from some other source, such as a name in MAX, that you select. Use this value to identify all instances of a particular switch type. Switches of the same type typically have the same session data type and same driver API.

TSM 2013 Examples

TSM includes the following new examples:

- **Asynchronous Analysis**—Demonstrates a test program that must perform lengthy analysis on data acquired from an instrument. The test program performs the analysis in an asynchronous thread.
- **Grading**—Demonstrates how to use the Set and Lock Bin step to grade DUTs based on different test criteria.
- **Multisite Simple Flow - No Hardware**—Demonstrates a multisite test program that uses LabVIEW code modules with simulated test results.
- **Switching**—Demonstrates how to use switching tools in the pin map to share an instrument across multiple sites.

Additional Enhancements in TSM 2013

TSM includes the following features and enhancements:

- Semiconductor Module Menu and Toolbar—Use this menu and related toolbar to edit test program and related files, to configure station settings and lot settings, to import or export test limits, and to disable TSM.
- Semiconductor Module Run-Time Error dialog box—Use this improved dialog box to specify how to handle run-time errors that occur in an execution that uses a process model if TSM is enabled. The dialog box contains a description of the error and the step, sequence, and sequence file where the error occurred. The dialog box includes an error code only for TestStand errors unrelated to TSM.
- Deploying TSM test systems—Use the TestStand Deployment Utility to deploy a test system. Launch the deployment utility and click **Drivers and Components** on the Installer Options tab to launch the Drivers and Components dialog box, in which you can include the TSM components as part of the installer you build.
- **Station options specify number of sites**—Use the **Require at Least N Sites in Pin Map** and the **Require at Most N Sites in Pin Map** options on the General tab of the Configure Station Settings dialog box to specify that a pin map must have at least **N** sites or at most **N** sites to run. Use the `AllowMorePinMapSitesThanTestSockets` and the `AllowFewerPinMapSitesThanTestSockets` properties of the NI_SemiconductorModule_StationSettings data type to specify whether TSM

runs a sequence file with a pin map with more or fewer sites than the number of test sockets configured in the model options.

TSM Licensing Options

After you install TSM, you must use the NI Activation Wizard to activate the software or initiate the evaluation period for the software. NI offers a variety of licenses for the different ways you can use TSM in development and deployment applications. You can select from the following types of licenses:

- NI TestStand Semiconductor Module Evaluation Package
- NI TestStand Semiconductor Module Development License (783522-35)
- NI TestStand Semiconductor Module Debug Deployment Environment License (779991-35)

TSM Operator Interface Improvements

The TSM operator interface includes general cosmetic improvements and the following functional improvements:

- Displays information about the lot and station settings.
- Displays generated report at execution completion or during test execution.
- Generates a mid-lot summary report.
- Automatically sorts bin table by highest count.

Documentation Updates

This help file now includes content on the following topics:

- Multisite Programming Techniques
- Improving Test System Performance

NI TestStand 2013 Semiconductor Module Compatibility Issues

The NI TestStand 2013 Semiconductor Module (TSM) introduces the following behavior changes between version 2012/2012 R2 and version 2013:

- TSM now installs the pin map VIs into `<vi.lib>/NI_TestStand_SemiconductorModule` instead of `<vi.lib>/addons/NI_TestStand_Se`

`miconductorModule`. The VIs now appear on the **TestStand Semiconductor Module»Pin Map** Functions palette instead of on the **Addons»Pin Map** Functions palette.

When you open test code VIs in LabVIEW, LabVIEW searches for the new location and updates the test code VIs automatically. To avoid this delay, mass compile the test code VIs by selecting **Tools»Advanced»Mass Compile** in LabVIEW.

- The following VIs are now deprecated. NI supports these VIs but recommends that you update files to reflect these changes to ensure compatibility with future versions of TSM.

Deprecated VI	Preferred VI
Create Multisite Digital Waveforms (Deprecated)	Create Multisite Digital Waveforms
Get Session And Channel Index (Deprecated)	Get Session And Channel Index
Pins to NI-HSDIO Channel Masks (Deprecated)	Pins to NI-HSDIO Channel Masks
Publish Data (Deprecated)	Publish Data
Rearrange Multisite Digital Waveforms (Deprecated)	Rearrange Multisite Digital Waveforms

- The NI TestStand 2013 Semiconductor Module changes the structure of the data types that represent lot settings and station settings. The new data types are not compatible with the types from the NI TestStand 2012/2012 R2 Semiconductor Module. You must update test programs in the following ways to account for the new data types:
  - When you open a sequence file saved with the NI TestStand 2012/2012 R2 Semiconductor Module data types for lot settings or station settings, variables that use those data types are automatically updated, and existing property values are lost. To retrieve the property values from a sequence file after you install the NI TestStand 2013 Semiconductor Module, select **Semiconductor Module»Disable Semiconductor Module** in the TestStand Sequence Editor before you open the sequence file. After you copy the values from station settings and lot settings variables, close the sequence file and re-enable TSM.

- You must update expressions that refer to properties in the NI_SemiconductorModule_LotSettings or NI_SemiconductorModule_StationSettings data types to avoid run-time errors, as the following table describes:

Property Type	How to Update
Standard properties	Change `Settings.` **PropertyName** to `Settings.Standard.` **PropertyName**
Custom properties	Add the properties to the NI_SemiconductorModule_CustomLotSettings or NI_SemiconductorModule_CustomStationSettings data types and remove any steps in sequences that use the TestStand API to create custom properties.

- TSM introduces a new template sequence file for you to use to create TSM callback sequences. Instead of copying callback sequences from the `SemiconductorModuleCallbacks.seq` sequence file in the `<TestStand>\Components\Callbacks\NI_SemiconductorModule` directory as recommended in previous versions of TSM, you must now copy the callback sequences from the `SemiconductorModuleCallbacks.seq` sequence file in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory. Do not copy sequences from any sequence files other than the sequence files in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory because only those sequence files use the current NI_SemiconductorModule_LotSettings or NI_SemiconductorModule_StationSettings data types.
- Some NI TestStand 2013 Semiconductor Module filenames and file paths have changed from the NI TestStand 2012/2012 R2 Semiconductor Module, as the following table describes:

2012/2012 R2 Name	2012/2012 R2 Path	2013 Name	2013 Path
`NI_SemiconductorModuleHandlerDriver_Template.seq`	`<TestStand>\Components\Modules\NI_SemiconductorModule`	`HandlerDriver.seq`	`<TestStand>\Components\Modules\NI_Semiconduct`

			orModule\Templates
NI_SemiconductorModuleInlineQAAlgorithm_Template.seq	<TestStand>\Components\Modules\NI_SemiconductorModule	InlineQAAlgorithm.seq	<TestStand>\Components\Modules\NI_SemiconductorModule\Templates

- The Semiconductor Multi Test step Options tab no longer includes options for specifying or editing pin map or bin definitions files. The Set and Lock Bin step tab no longer includes options for specifying or editing the bin definitions file. Use the Edit Test Program dialog box to specify and edit the pin map and bin definitions files.

What's New in the NI TestStand 2012 R2 Semiconductor Module

The NI TestStand 2012 R2 Semiconductor Module introduces new features. Some compatibility issues might exist as a result of changes in the NI TestStand 2012 R2 Semiconductor Module.

NI TestStand 2012 R2 Semiconductor Module New Features

The following list describes the new features in the NI TestStand 2012 R2 Semiconductor Module (TSM) and other changes since the NI TestStand 2012 Semiconductor Module.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

LabVIEW VIs and Palettes

TSM adds the following new VIs:

- **Create Multisite Digital Waveforms**—Creates multisite digital waveforms based on the pin map.
- **Rearrange Multisite Digital Waveforms**—Rearranges multisite waveforms read from NI-HSDIO instruments into per-site digital waveforms.
- **Get Pin Names**—Returns all DUT and system pins.

- **Get Session And Channel Index VI**—Returns the index of the channel group and channel that corresponds to a pin query. Use this VI to access an individual pin when you take a measurement across multiple instruments and pins. When you call a pin query VI, such as the Pins to NI-HSDIO Sessions VI, the VI returns an array of sessions and a channel list. Use the Get Session and Channel Index VI to identify which session and which channel refers to a pin and site number you specify.

TSM adds the following new subpalettes:

- **NI-HSDIO subpalette**—Use this subpalette to access the NI-HSDIO VIs and the Multisite Digital Waveform VIs.
- **NI-DCPower subpalette**—Use this subpalette to access the NI-DCPower VIs.

New Reports and Data Logs

You can generate the following new types of TSM reports and data logs. Enable and configure the corresponding TSM result processing plug-in to generate the report or data log.

- Lot Summary Report—Text file that contains a summary of the semiconductor test results for the current lot of DUTs (test lot).
- Test Results Log—Human-readable text file that contains the measurement values and test limits for each test that executes on each site.

You can customize the filenames and locations in which the TSM result processing plug-ins create report and data log files by modifying a copy of the GetReportFileName callback.

Customizing the STDF PRR PartID Field Value

The STDF result processing plug-in sets the PartID field in the Part Results Record (PRR) of the STDF version 4 specification by using the value of the `SerialNumber` property on the UUT data type. By default, TSM automatically assigns sequential numeric values to the `SerialNumber` property, which results in unique PartID field values for each PRR.

The NI_SemiconductorModule_StationSettings data type now includes a `GenerateUniquePartIds` property that specifies whether TSM generates unique values

for the PartId field of PRRs in the STDF log file. Set this value to `False` to generate unique PartID values using a custom algorithm you implement.

LotTestingComplete Callback

TSM calls the `LotTestingComplete` callback sequence to perform tasks when a lot completes testing, such as sending generated reports to a central server or displaying a message on the tester to indicate that the tester is idle.

The default implementation of the `LotTestingComplete` callback sequence is empty. You can override this callback to customize the tasks to perform when a lot completes testing. TSM calls the LotTestingComplete callback after all other process model plug-ins complete execution.

Improved Lot and Station Settings Configuration Dialog Boxes

The default Configure Lot Settings dialog box includes the following new fields:

- Device Name
- Lot Number
- Estimated Lot Size
- Test Flow
- Test Temp
- Operator ID

The Configure Station Settings dialog box includes the following new fields:

- Test Failure Mode
- Number of Sites
- Additional buttons that launch the standard TestStand Station Options, Search Directories, LabVIEW Adapter, and Result Processing dialog boxes

Operator Interface Enhancements

The TSM default operator interface now displays the following new information:

- Average test times, such as Socket Time and Cycle Time, calculated by averaging the last 10 runs of a DUT.

- Status of the tester with information for operators about how to proceed when testing a lot.
- Sites that are actively testing DUTs, DUTs that have recently failed on a given site, or sites that are waiting for DUTs to test.
- Inline QA data, when enabled, such as the number of DUTs that passed and failed during inline QA testing.
- Errors that occur during execution and that appear in an error log file.

NI TestStand 2012 R2 Semiconductor Module Compatibility Issues

The NI TestStand 2012 R2 Semiconductor Module introduces the following behavior changes between version 2012 and version 2012 R2:

Changes to Pin Map XML Structure

The `InstrumentTypeId` attribute of the `<Instrument>` element must now contain at least one character. If you previously used an empty string for `InstrumentTypeId`, the XML file no longer validates. Enter at least one character for the attribute and update any code modules that refer to the `InstrumentTypeId` attribute.

GetSTDFFileName Callback Removed

The STDF Log result processor model plug-in no longer uses the `GetSTDFFileName` callback sequence to determine the name of the STDF log file. Use the GetReportFileName callback sequence instead.

Customizing the Configure Lot Settings and Configure Station Settings Dialog Boxes

You must now complete the following additional steps to customize the behavior for obtaining lot settings and station settings.

- Copy the `<TestStand>\Components\Callbacks\NI_SemiconductorModule\LotSettingsDialogs.lvlibp` and `StationSettingsDialogs.lvlibp` LabVIEW packed project library files to the corresponding `<TestStand Public>\Components\Callbacks\NI_Semiconduct`

orModule\LotSettingsDialogs.lvlibp and StationSettingsDialogs.lvlibp.

- Copy the contents of <TestStand>\Components\Callbacks\NI_SemiconductorModule\Source\LotSettingsDialogs and StationSettingsDialogs directories to the corresponding <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\Source\LotSettingsDialogs and StationSettingsDialogs directories and make changes to the copy of the LabVIEW projects.
- Rebuild the packed project library build specifications in the projects to update the copies of the LabVIEW packed project libraries.

Getting Started with TSM

Use TestStand and TSM with other NI development tools to build, debug, customize, and deploy semiconductor characterization and production test systems.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

▾ Brief Tour of TSM

Note You must have the LabVIEW Development System installed to use this example.

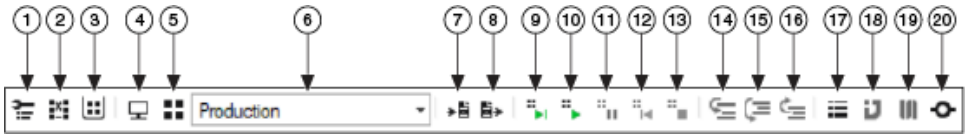
1. From the TestStand Sequence Editor, open Getting Started with Semiconductor Module.seq. Open the file from the <TestStand Public>\Examples\NI_SemiconductorModule\Getting Started with Semiconductor Module\LabVIEW directory.

   <TestStand Public>\Examples\NI_SemiconductorModule\Getting Started with Semiconductor Module\LabVIEW\Getting Started with Semiconductor Module.seq

   The Getting Started with Semiconductor Module.seq sequence file is a simple example semiconductor test program that

demonstrates a multisite test program that uses LabVIEW code modules with simulated test results. The example is configured to test up to four DUTs in parallel, each on a separate test site.

2. Review the following TSM toolbar buttons to use to control execution and view lot statistics while executing and debugging a sequence.



1. Edit Test Program 2. Edit Pin Map File 3. Edit Bin Definitions File 4. Configure Station 5. Configure Lot 6. Active Configuration 7. Import Test Limits into	8. Export Test Limits from 9. Single Test 10. Start/Resume Lot 11. Pause 12. Retest 13. End Lot 14. Step Into	15. Step Over 16. Step Out 17. Show Lot Statistics Viewer 18. Launch Digital Pattern Editor 19. Launch InstrumentStudio 20. Enable/Disable Offline Mode

3. Click the **Start/Resume Lot** button to begin testing. Each Execution window represents a test site. The sequence editor traces each step during execution.

Note Depending on the current sequence editor settings, when you click the **Start/Resume Lot** button, TestStand might display the **Found Analysis Errors** dialog box to indicate that errors exist in the sequence file. Click the **Continue Execution** button to ignore these errors because the sequence file adjusts certain settings at run time to fix these errors.

4. Click the **Show Lot Statistics Viewer** button to view per-site binning information. The Lot Statistics Viewer window includes the same buttons as on the TSM toolbar for controlling execution.
5. Click the **End Lot** button to stop testing.

6. Close the Lot Statistics Viewer window.
7. Press <Ctrl-D> to close the Execution windows.

Debugging LabVIEW Steps

1. In the `Getting Started with Semiconductor Module.seq`, click in the blank column to the left of the Leakage step to set a breakpoint ●.
2. Click the **Start/Resume Lot** button to begin testing. Each Execution window pauses when it reaches the breakpoint at the Leakage step. The background color of the Execution window changes to yellow to indicate that the execution is paused.
3. Click the **Step Into** button on the TSM toolbar to transfer execution to the LabVIEW Development System, which suspends within the Leakage VI the Leakage step calls.

   You can now use the built-in LabVIEW debugging tools.
4. In LabVIEW, click the **Run** button and then click the **Return to Caller** button to return execution to the sequence editor.
5. Click the **End Lot** button to stop testing.

Where to Go Next

- Explore the components of a test program.
- Complete the Exploring a Basic Semiconductor Test Program tutorial.
- Review the Accelerometer example, located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer` directory.
- Use the TestStand and TSM example programs, located in the `<TestStand Public>\Examples` directory, as a starting point for applications you create.
- Refer to the NI STS Technical Support Community on `ni.com` for information about TSM custom instruments for instrument drivers, custom debug panels, and custom handler/prober drivers. You can work directly with NI services personnel contracted on your project or contact `stssupport@ni.com` to request to be added to the NI STS Technical Support Community.

Related TestStand Resources

Familiarize yourself with TestStand in the following ways:

- Complete the tutorials in the **Getting Started with TestStand** manual.
- Review the following information in the **TestStand help** and familiarize yourself with the organization of the help file
  - Guide to TestStand Documentation
  - TestStand Components
  - General Test Executive Concepts
  - TestStand Building Blocks

Overview of Test Program Components (TSM)

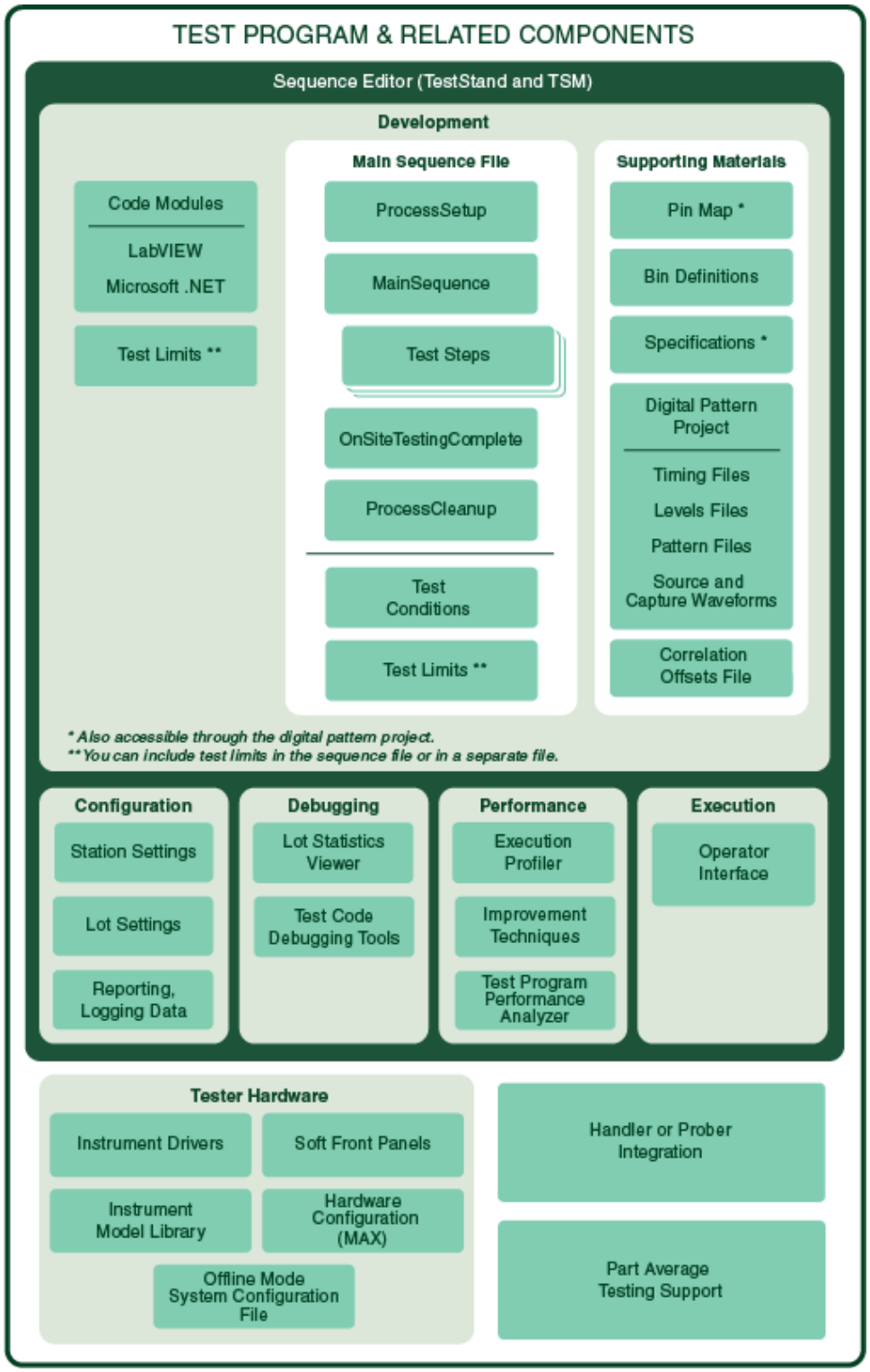
A semiconductor test program can include a pin map file, a main sequence file, subordinate sequence files, code modules, specifications, timing files, levels files, pattern files, source and capture waveforms, test limits files, a bin definitions file, and configurations. Use the Test Program Editor to complete the following tasks:

- Specify the pin map, bin definitions, specifications, digital pattern project, and test limits files
- Create and edit test program configurations
- Configure other settings for the test program

Select **Semiconductor Module»Edit Test Program: <filename>** or click the **Edit Test Program: <filename>** button on the TSM toolbar to launch the Test Program Editor for the sequence file.

Use the TestStand Sequence Editor to complete test program development, configuration, debugging, and execution tasks.

Click the items in the following figure for more information about the components of test programs and test stations.


TEST PROGRAM & RELATED COMPONENTS
Sequence Editor (TestStand and TSM)
Development
Code Modules
LabVIEW
Microsoft .NET
Test Limits **
Main Sequence File
ProcessSetup
MainSequence
Test Steps
OnSiteTestingComplete
ProcessCleanup
Test Conditions
Test Limits **
Supporting Materials
Pin Map *
Bin Definitions
Specifications *
Digital Pattern Project
Timing Files
Levels Files
Pattern Files
Source and Capture Waveforms
Correlation Offsets File
* Also accessible through the digital pattern project.
** You can include test limits in the sequence file or in a separate file.
Configuration
Station Settings
Lot Settings
Reporting, Logging Data
Debugging
Lot Statistics Viewer
Test Code Debugging Tools
Performance
Execution Profiler
Improvement Techniques
Test Program Performance Analyzer
Execution
Operator Interface
Tester Hardware
Instrument Drivers
Soft Front Panels
Instrument Model Library
Hardware Configuration (MAX)
Offline Mode System Configuration File
Handler or Prober Integration
Part Average Testing Support

TestStand Sequence Editor

The TestStand Sequence Editor is the development environment in which you create, edit, execute, and debug sequences and the tests sequences call.

Complete the following steps to launch the sequence editor.

1. (Windows 8.1/8) Click the **NI Launcher** tile on the Start screen and select **TestStand»TestStand Sequence Editor**.

   (Windows 7) Select **Start»All Programs»National Instruments»TestStand»TestStand Sequence Editor**.

   (Windows 10) Select **Start» NI TestStand**.

   The sequence editor launches the main window and the Login dialog box.
2. Use the default user name, `administrator`, in the User Name ring control. Leave the Password field empty. You can use the TestStand User Manager to customize user settings and permissions.
3. Click **OK**.

Back to Overview

Pin Map

A pin map defines the instrumentation on the tester, defines the pins on the DUT, and defines how the DUT pins are connected to the tester instrumentation for each test site. Use the Pin Map Editor to view, create, modify, and save pin map files. The pin map file also serves as the channel map file.

Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to launch the Pin Map Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Pin Map** in the Test Program Editor to launch the Pin Map panel. Click the **Open file for edit** button to launch the Pin Map Editor.

Back to Overview

Bin Definitions File

A bin definitions file defines the hardware bins and software bins, defines how the software bins relate to hardware bins, and defines the default software bins for the test program main sequence file. Use the Bin Definitions Editor to view, create, modify, and save bin definitions files.

Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar to launch the Bin Definitions Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Bin Definitions** in the Test Program Editor to launch the Bin Definitions panel. Click the **Open file for edit** button to launch the Bin Definitions Editor.

Back to Overview

Specifications Files

Specifications files define a set of symbols, or variables, and associated numeric values of DUT attributes that you can reference in test program code modules instead of using constants to set testing specifications. You can modify specifications files to set new values without changing test program files.

Select **Semiconductor Module»Edit Test Program** and select **Specifications Files** in the Test Program Editor to specify one or more specifications files to load in the test program.

Back to Overview

Digital Pattern Project

Use the Digital Pattern Editor with a Digital Pattern Instrument and the NI-Digital Pattern Driver software for digital testing of semiconductors, or devices under test (DUTs). Use digital pattern project files to organize and access pattern files and the following types of digital configuration files:

- Pin and channel maps (`.pinmap`)
- Specifications (`.specs`)

- Digital timing (`.digitiming`)
- Digital levels (`.digilevels`)
- Digital patterns (`.digipat`)
- Source waveforms (`.tdms`)
- Capture waveform configurations (`.digicapture`)

Back to Overview

Timing Files

Timing files contain configuration components of digital pattern time sets, which define the behavior of a digital signal on a pin for a particular cycle. Timing files also include the format and edge placement that shape the digital waveform on a per-pin basis for a digital pattern instrument. Edit timing files in the Digital Pattern Editor.

Back to Overview

Levels Files

Levels files contain voltage and current levels to drive and compare for digital pins and pin groups connected to an NI digital pattern instrument and for pins and pin groups connected to an NI-DCPower instrument. Edit levels files in the Digital Pattern Editor.

Back to Overview

Pattern Files

Pattern files contain a collection of vectors, or instructions, to execute on an NI digital pattern instrument. Components of the binary pattern file include time sets, labels, opcodes, vector numbers, pin state data that indicates drives and compares, and comments for each vector. Edit pattern files in the Digital Pattern Editor.

Back to Overview

Source and Capture Waveforms

You can source or capture a variable waveform that is not defined at compile time on an NI digital pattern instrument. Edit source waveform and capture waveform configuration files in the Digital Pattern Editor.

Back to Overview

Correlation Offsets File

You can apply correlation offset values to test results on a per-site basis at run time before evaluating the test result data against limits. Use the Load Correlation Offsets Step and associated edit tab to load and apply a correlation offset file.

Back to Overview

Test Program Configurations

Test program configurations define values for conditions that a test program can reference at run time and the test limits file that loads before running a test lot. A test program can use multiple configurations to implement multiple test flows using the same sequences and code modules. For example, you can create configurations for Hot and Cold flows or for QA and Production lots.

Select **Semiconductor Module»Edit Test Program** and then select **Configuration Definition** in the Test Program Editor to define configuration settings.

Back to Overview

Test Limits Files

Test limits files define test limits the test program loads before running a test lot. The test program replaces test limits in test steps in the sequence file with those specified in the test limits file. You can embed test limits in the sequence file to prevent viewing or tampering with the limits.

Select **Semiconductor Module»Edit Test Program** and select **Test Limits Files** in the Test Program Editor to specify one or more test limits files to make available to the test program configurations. The test program configuration specifies the test limits file that loads before running a test lot.

You can create a test limits file by selecting **Semiconductor Module»Export Test Limits from** or by clicking the **Export Test Limits from** button on the TSM toolbar to export test limits from a sequence file into a tab-delimited test limits text file. You can import a test limits file by selecting **Semiconductor Module»Import Test Limits to** or by clicking the **Import Test Limits to** button on the TSM toolbar to import test limits from a tab-delimited test limits text file into a sequence file. When you import test limits from a text file, you can update limits in matching tests or replace all tests in matching steps.

Back to Overview

Test Conditions

Test conditions specify historical information, descriptive information, such as DUT numbers or package types, and conditions under which to test the DUTs, such as temperature or voltage. The test program can use test conditions to determine how to execute tests. For example, test conditions might dictate which steps execute, what temperature to apply to a DUT, what voltage to use, and so on.

Select **Semiconductor Module»Edit Test Program** and select an option in the **Configurations** list of the Test Program Editor to edit configuration settings.

Back to Overview

Main Sequence File

The main sequence file contains the sequences that define the test flow by specifying the test steps to execute and the order in which to execute them. The sequence file contains one main sequence named `MainSequence` and can optionally include one or more subsequences with corresponding test steps. You can use multiple sequences in a test program to keep the test code modular and organized.

The `ProcessSetup` and `ProcessCleanup` sequences are special sequences that TestStand calls at certain times. TestStand calls `ProcessSetup` once before starting execution and calls `ProcessCleanup` after execution completes. Initialization and cleanup of instrumentation typically occurs within these sequences.

Back to Overview

Test Steps

Test steps call test code in code modules that control the instrumentation on the tester. The test steps perform tests by comparing measurement values obtained by the code modules to test limits stored on the step and assign a bin to the DUT if the comparison fails. You can also assign a bin to the DUT when the comparison fails. Test steps are instances of the Semiconductor Multi Test step type, in which you also define test numbers and names.

Back to Overview

OnSiteTestingComplete

Use the `OnSiteTestingComplete` callback sequence to perform actions on a DUT or on instruments after all DUT tests have completed and TSM has assigned a bin to the DUT.

To use the `OnSiteTestingComplete` callback, add a sequence with no parameters to the test program sequence file and name the new sequence `OnSiteTestingComplete`. TSM calls the sequence after the `MainSequence` sequence, after all PAT tests complete, and after TSM assigns a bin to the DUT. Use the Get Test Information step in the callback sequence to determine the bin assigned to the DUT.

You cannot use Semiconductor Multi Test steps in the `OnSiteTestingComplete` callback because all tests must execute before TSM assigns a bin to the DUT and before TSM calls the callback. Use the Semiconductor Action step to perform operations with the instruments and DUT.

You cannot change the bin assigned to a DUT in the `OnSiteTestingComplete` callback. You cannot use the Set and Lock Bin step in the `OnSiteTestingComplete` callback.

You can use the Cleanup step group of the `MainSequence` sequence to perform similar actions as the `OnSiteTestingComplete` callback if you do not need to know the bin assigned to the DUT to perform the actions. If your test program uses part average testing, you should precede the actions in the Cleanup step group with a Perform Part Average Testing step, to ensure that they run after all part average testing is completed. If you don't use a Perform Part Average Testing step, TSM performs part average testing after executing all step groups of the `MainSequence` sequence.

Back to Overview

Code Modules

Use LabVIEW or .NET to create, edit, and debug test code in code modules to control the instrumentation on the tester, take measurements from the DUT, and pass the measurement values back to the test step. Code modules are program modules, such as a LabVIEW VI or a Microsoft Windows DLL, that contain one or more functions that perform a specific test or other action.

Back to Overview

Station Settings

You can specify test station configuration options for the tester, such as handler configuration or data logging preferences, that apply to all test lots and that persist during restart and shutdown operations. The test program can use station information to determine how to execute tests. For example, station settings might specify the type of handler to use with the test program, which reports to generate, or whether the test station performs inline quality assurance testing. When a test station is reconfigured, such as to specify a different handler or to change the functionality of the tester, the station settings must be updated to account for the changes. You can customize how TSM obtains and processes the settings.

Select **Semiconductor Module»Configure Station** in the TestStand Sequence Editor or click the **Configure Station** button in the default TSM operator interface to launch the Configure Station Settings dialog box.

Back to Overview

Lot Settings

The test program can use lot information to determine how to execute tests. For example, lot settings might dictate which steps execute, what temperature to apply to a DUT, what voltage to use, and so on. You can customize how TSM obtains the settings.

Select **Semiconductor Module»Configure Lot** in the TestStand Sequence Editor or click the **Configure Lot** button in the default TSM operator interface to launch the Configure Lot Settings dialog box.

Back to Overview

Reporting and Data Logging

You can generate TSM reports and data logs, such as Standard Test Data Format (STDF) log files, Lot Summary Reports, and Test Results Logs. Enable and configure the corresponding TSM result processing plug-in to generate the report or data log. You can customize the destination directory and filename of the report or data log file.

Back to Overview

Lot Statistics Viewer

The Lot Statistics Viewer window provides a way to view lot statistics, including per-site bin counts, while running or debugging a sequence in the sequence editor. You can also control test program execution in the Lot Statistics Viewer.

Select **Semiconductor Module»Show Lot Statistics Viewer** or click the **Show Lot Statistics Viewer** button on the TSM toolbar to launch the Lot Statistics Viewer window.

Back to Overview

Test Code Debugging Tools

TestStand includes several tools for debugging sequences and related components in a TestStand test program and in TSM test programs. Additionally, the TestStand Sequence Editor integrates with supported application development environments to debug code modules.

Back to Overview

TestStand Execution Profiler

Use the Execution Profiler to view and record duration of steps, code modules, and other resources a multithreaded TestStand system uses over a period of time.

In an effort to improve test time performance, you can optimize test time by identifying parts of the test program that take longest to execute or by identifying what shared tester resources cause throughput bottlenecks.

Back to Overview

Test Program Performance Analyzer

Use the Test Program Performance Analyzer to view data TSM generates when you measure the performance of a test program. You can filter, graph, compare, and save the data in various ways to identify performance issues in a test program.

Back to Overview

Operator Interfaces

Use TSM default operator interface applications to execute test sequences on a test station. The operator interface source code is available in LabVIEW and C#. You can fully customize them to meet specific needs.

Back to Overview

Instrument Drivers

NI provides instrument drivers to configure, customize, and implement your instrument control applications. TSM has native support for multiple NI instrument drivers. You can integrate other NI or third-party drivers into the multisite pin map by using custom instruments. You can download TSM custom instruments for more drivers from the NI STS Technical Support Community on `ni.com`. You can work directly with NI services personnel contracted on your project or contact `stssupport@ni.com` to request to be added to the NI STS Technical Support Community.

Note The NI TestStand 2016 Semiconductor Module natively supports NI digital pattern instruments that use the NI-Digital Pattern Driver and legacy digital waveform instruments that use the NI-HSDIO driver, such as the PXIe-6556. Use the TSM Code Module API that corresponds to the type of digital instrument the test system includes.

Back to Overview

Soft Front Panels

Most NI modular instruments include soft front panels (SFP) to allow you to quickly configure, troubleshoot, or debug your instrument or DUT. Launch the soft front panels from MAX by selecting **Tools»Soft Front Panels**. TSM also provides tools to create custom debug panels in LabVIEW or C#. You can launch custom debug panels when debugging a test program, and you can use them to configure and control multiple instrument drivers.

You can also use InstrumentStudio, a software-based front panel application, to monitor, control, and record measurements from supported devices.

Before you create a custom debug panel, search the NI STS Technical Support Community on `ni.com` for an existing one. You can work directly with NI services personnel contracted on your project or contact `stssupport@ni.com` to request to be added to the NI STS Technical Support Community.

Back to Overview

Offline Mode System Configuration File

The Offline Mode system configuration file defines the instruments on a specific STS.

Back to Overview

Instrument Model Library

The Instrument Model Library is a collection of XML files that describe instruments. Instrument model description files include general information, details for connection components (channels, ports, resources), and configuration properties of the instrument and its resources.

Back to Overview

Measurement & Automation Explorer

The TSM pin map defines the instruments required for a test. Measurement & Automation Explorer (MAX) helps you configure the instrument connected to the system. For a test program to execute properly, the instrument names in the pin map must correspond with instrument names configured in the system.

MAX helps you complete the following tasks:

- Configure NI hardware and software and third-party IVI hardware and software
- View and edit instruments names in the system
- Create and edit channels, tasks, interfaces, scales, and virtual instruments

- Execute system diagnostics and run soft front panels
- Update NI software

(Windows 8.1/8) Click the **NI Launcher** tile on the Start screen and select **NI MAX** to launch MAX. (Windows 7) Select **Start»NI MAX** to launch MAX.

Back to Overview

Handler or Prober Integration

The TSM handler/prober driver plug-in architecture enables you to write and enable handler/prober drivers. A handler/prober driver contains entry point sequences that TSM calls during execution to accomplish handler-related or prober-related tasks.

Use the NI Built-in Simulated Handler Driver to simulate handler functionality without requiring access to a real handler.

Custom handler/prober drivers are available from the NI STS Technical Support Community on `ni.com`. You can work directly with NI services personnel contracted on your project or contact `stssupport@ni.com` to request to be added to the NI STS Technical Support Community.

Back to Overview

Part Average Testing Support

Part average testing (PAT) is a method based on statistical analysis to identify and fail parts that have characteristics significantly outside the normal distribution of other parts in the same lot. TSM does not install a default implementation of part average testing. You must use the TSM PAT plug-in architecture to customize and perform part average testing with TSM. TSM PAT plug-ins include a required PAT callback sequence file and corresponding code modules. The PAT callback sequence file contains PAT entry point sequences that TSM calls during execution to accomplish part average testing.

Back to Overview

See Also

Getting Started with TSM

Tutorial: Exploring a Basic Semiconductor Test Program

Use this LabVIEW or .NET tutorial to explore and configure components of a test program after you tour the features available in TSM.

Tutorial: Exploring a Basic Semiconductor Test Program with LabVIEW

Complete the following steps to open an existing sequence file and configure it to create a basic semiconductor test program.

Note Completed solution files are located in the `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\LabVIEW\Solution` directory.

1. Open `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\LabVIEW\Basic Test Program.seq`. This sequence file contains the following sequences:
   - **MainSequence**—Contains a single test step that performs a continuity test on all pins.
   - **ProcessSetup**—Simulates instrument initialization.
   - **ProcessCleanup**—Simulates instrument clean up.
2. Complete the following steps to specify a pin map file to define the instrumentation on the tester, define the pins on the DUT, and define how the DUT pins are connected to the tester instrumentation for each test site.
   a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq** or click the **Edit Test Program: Basic Test Program.seq** button on the TSM toolbar to launch the Test Program Editor for the sequence file.
   b. Select the Pin Map panel and enter `Basic Test Program.pinmap` in the **Pin Map File Path** control. The filename you enter is a relative

path from the sequence file to the pin map file, `Basic Test Program.pinmap`, which is located in the same directory as the sequence file. Click the **Open file for edit** button located to the right of the Pin Map File Path display to open the `Basic Test Program.pinmap` file in the Pin Map Editor. Review the pin map and click **OK** to close the pin map editor.

c. Click **OK** to close the Test Program Editor.

After you specify a pin map file, select the `MainSequence` sequence in the sequence file. This sequence contains the Continuity Test step, which is an instance of the Semiconductor Multi Test step type and uses the LabVIEW Adapter. Select the Continuity Test step and select the Tests tab of the Step Settings pane. You can use the Tests tab to define tests for individual pins or pin groups, and you can write code modules that refer to pin or pin group names without needing to know how each pin is connected to an instrument.

3. Complete the following steps to specify a bin definitions file to define the hardware bins and software bins, define how the software bins relate to hardware bins, and define the default software bins in the test program.

   a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq**.

   b. Select the Bin Definitions panel and enter `Basic Test Program.bins` in the Bin Definitions File Path control. The filename you enter is a relative path from the sequence file to the bin definitions file, `Basic Test Program.bins`, which is located in the same directory as the sequence file. Click the **Open file for edit** button located to the right of the Bin Definitions File Path display to open the `Basic Test Program.bins` file in the Bin Definitions Editor. Review the bin definitions and click **OK** to close the Bin Definitions Editor.

   c. Click **OK** to close the Test Program Editor.

After you specify a bin definitions file, you can use the Tests tab of the Step Settings pane to define a software bin for each test.

4. Complete the following steps to create a test program configuration and specify a test limits file to load limits values into Semiconductor Multi Test step tests at run time. When you use this technique, you can use the same test program with multiple configurations that each specify different limits for the tests.
    a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq**.
    b. Select the Test Limits Files panel and click the **Add Test Limits File** button to add a new file reference to the test program.
    c. Enter `Production Limits` as the name to identify the new test limits file in the test program.
    d. Enter `ProductionLimits.txt` in the **Test Limits File Path** control. The filename you enter is a relative path from the sequence file to the test limits file, `ProductionLimits.txt`, which is located in the same directory as the sequence file.
    e. Select the Configurations panel and click the **Add Configuration** button to add a new configuration to the test program.
    f. Enter `Production` as the name of the configuration.
    g. Select the newly created **Production** panel. Each configuration you specify uses a corresponding Configuration panel that contains a table of the test conditions in the test program and fields for configuring the test limits file for the configuration.
    h. Select **Production Limits** in the **Test Limits File** control.
5. Complete the following steps to add a test condition to the test program and specify a value for the test condition in the Production configuration.
    a. Select the **Configuration Definition** panel and click the **Add Condition** button to add a new test condition to the test program.
    b. Select **TestFlowId** from the **Standard Condition** option and click the **Add** button to add a new test condition for which each configuration can provide a unique value.

c. Select the **Production** panel and enter `Production` for the value of the TestFlowId test condition in the table.

d. Click **OK** to close the Test Program Editor.

6. Complete the following steps to obtain the value of the TestFlowId test condition in the test program.

   a. Select the `ProcessSetup` sequence.

   b. Add a Get Test Information step after the Open Sessions step.

   c. On the Step Settings pane, click the **Get Test Information** tab.

   d. Enter `TestFlowId` for the **Name** in the first row of the table.

   e. Set the **Destination Expression** to `Locals.Flow`.

   f. Highlight and right-click `Locals.Flow` and select **Create "Locals.Flow"»String** from the context menu to create a new local variable.

   g. Add a Message Popup step after the Get Test Information step in the Setup step group.

   h. On the Step Settings pane, click the **Text and Buttons** tab.

   i. Set the Message Expression to `"Starting test flow: " + Locals.Flow`.

7. Complete the following steps to specify a code module for the Continuity Test step.

   a. Select the `MainSequence` sequence and the Continuity Test step.

   b. On the Step Settings pane, click the **Module** tab.

   c. Click the **Browse for VI** button located to the right of the **VI Path** control.

   d. Browse to `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\LabVIEW\Code Modules\Continuity Test.vi` and click **Open**. Select the **Use a relative path for the file you selected** option when prompted and click **OK**.

   e. In the VI Parameter Table, configure the following parameter values:

Parameter	Value
Semiconductor Module Context	`Step.SemiconductorModuleContext`
Pins	`{"AllPins"}`

8. Save the sequence file.
9. Complete the following steps to configure the Built-in Simulated Handler Driver.
    a. Select **Semiconductor Module»Configure Station** or click the **Configure Station** button on the TSM toolbar to launch the Configure Station Settings dialog box.
    b. On the General tab of the Configure Station Settings dialog box, place a checkmark in the **Enable Handler/Prober Driver (Real or Simulated)** checkbox.
    c. Select **Built-in Simulated Handler Driver** from the **Handler/Prober Driver** drop-down menu.
    d. Click the **Configure Handler/Prober Driver** button to launch the Configure Built-in Simulated Handler dialog box.
    e. Set the **Number of DUTs to Test** option to `10` to specify how many DUTs the simulated lot contains and click **OK** to close the Configure Built-in Simulated Handler dialog box.
    f. Select the Advanced tab in the Configure Station Settings dialog box and click the **Result Processing** button to launch the Result Processing dialog box.
    g. Enable the **Debug Test Results Log** result processing plug-in.
    h. Disable the TestStand **Report** result processing plug-in.
    i. Enable the Display option for the **Lot Summary Report** result processing plug-in.
    j. Click **OK** to close the Result Processing dialog box, and click **OK** to close the Configure Station Settings dialog box.
10. Complete the following steps to configure a lot to run.

a. Select **Semiconductor Module»Configure Lot** or click the **Configure Lot** button on the TSM toolbar to launch the Configure Lot Settings dialog box.

b. Select **Production** in the **Test Program Configuration** option.

c. Click **OK** to close the Configure Lot Settings dialog box.

Note You can also use the **Active Configuration** drop-down menu on the TSM toolbar to set the configuration. TSM uses previously set values for the other lot settings that appear in the Configure Lot Settings dialog box.

11. Click the **Show Lot Statistics Viewer** button on the TSM toolbar to launch a floating window that shows the binning results of the test program execution. The execution control buttons, such as **Start/Resume Lot** and **End Lot** and the **Configuration** drop-down menu are also available in the Lot Statistics Viewer for convenience.
12. Click the **Start Lot** button on the TSM toolbar or in the Lot Statistics Viewer to execute the test program. The test program launches a dialog box that shows the value of the TestFlowId test condition. Click **OK** to close the dialog box. TestStand generates and displays the Lot Summary Report on the Report pane of the Execution window. Review the results of the lot.
13. Click the **Active Report** button to switch to the Debug Test Results Log and review the results of each test.

Note The Production configuration overrides the test limits the sequence specifies. The `ProductionLimits.txt` test limits file provides the test limits values to use for each test instead of the values entered in the Semiconductor Multi Test step.

14. Close the Execution and Sequence File windows.

See Also

Getting Started with TSM

Glossary

Tutorial: Importing Test Limits from a File

Tutorial: Exploring a Basic Semiconductor Test Program with .NET

Complete the following steps to open an existing sequence file and configure it to create a basic semiconductor test program.

Notes

- You must have NI-DCPower 15.1 or later installed, and you must have .NET 4.0 support for the NI-DCPower .NET Class Libraries 1.1 or later installed. If you do not have these components installed, refer to the Exploring a Basic Semiconductor Test Program with LabVIEW tutorial, which follows the same procedure but does not require these installations.
- Completed solution files are located in the `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\DotNET\Solution` directory.

1. Open `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\DotNET\Basic Test Program.seq`. This sequence file contains the following sequences:
   - **MainSequence**—Contains a single test step that performs a continuity test on all pins.
   - **ProcessSetup**—Simulates instrument initialization.
   - **ProcessCleanup**—Simulates instrument clean up.

2. Complete the following steps to specify a pin map file to define the instrumentation on the tester, define the pins on the DUT, and define how the DUT pins are connected to the tester instrumentation for each test site.

    a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq** or click the **Edit Test Program: Basic Test Program.seq** button on the TSM toolbar to launch the Test Program Editor for the sequence file.

    b. Select the Pin Map panel and enter `Basic Test Program.pinmap` in the **Pin Map File Path** control. The filename you enter is a relative path from the sequence file to the pin map file, `Basic Test Program.pinmap`, which is located in the same directory as the sequence file. Click the **Open file for edit** button located to the right of the Pin Map File Path display to open the `Basic Test Program.pinmap` file in the Pin Map Editor. Review the pin map and click **OK** to close the pin map editor.

    c. Click **OK** to close the Test Program Editor.

    After you specify a pin map file, select the `MainSequence` sequence in the sequence file. This sequence contains the Continuity Test step, which is an instance of the Semiconductor Multi Test step type and uses the .NET Adapter. Select the Continuity Test step and select the Tests tab of the Step Settings pane. You can use the Tests tab to define tests for individual pins or pin groups, and you can write code modules that refer to pin or pin group names without needing to know how each pin is connected to an instrument.

3. Complete the following steps to specify a bin definitions file to define the hardware bins and software bins, define how the software bins relate to hardware bins, and define the default software bins in the test program.

    a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq**.

    b. Select the Bin Definitions panel and enter `Basic Test Program.bins` in the Bin Definitions File Path control. The filename you enter is a relative path from the sequence file to the bin definitions file, `Basic T`

`est Program.bins`, which is located in the same directory as the sequence file. Click the **Open file for edit** button located to the right of the Bin Definitions File Path display to open the `Basic Test Program.bins` file in the Bin Definitions Editor. Review the bin definitions and click **OK** to close the Bin Definitions Editor.

c. Click **OK** to close the Test Program Editor.

After you specify a bin definitions file, you can use the Tests tab of the Step Settings pane to define a software bin for each test.

4. Complete the following steps to create a test program configuration and specify a test limits file to load limits values into Semiconductor Multi Test step tests at run time. When you use this technique, you can use the same test program with multiple configurations that each specify different limits for the tests.

   a. Select **Semiconductor Module»Edit Test Program: Basic Test Program.seq**.

   b. Select the Test Limits Files panel and click the **Add Test Limits File** button to add a new file reference to the test program.

   c. Enter `Production Limits` as the name to identify the new test limits file in the test program.

   d. Enter `ProductionLimits.txt` in the **Test Limits File Path** control. The filename you enter is a relative path from the sequence file to the test limits file, `ProductionLimits.txt`, which is located in the same directory as the sequence file.

   e. Select the Configurations panel and click the **Add Configuration** button to add a new configuration to the test program.

   f. Enter `Production` as the name of the configuration.

   g. Select the newly created **Production** panel. Each configuration you specify uses a corresponding Configuration panel that contains a table of the test conditions in the test program and fields for configuring the test limits file for the configuration.

   h. Select **Production Limits** in the **Test Limits File** control.

5. Complete the following steps to add a test condition to the test program and specify a value for the test condition in the Production configuration.
    a. Select the **Configuration Definition** panel and click the **Add Condition** button to add a new test condition to the test program.
    b. Select **TestFlowId** from the **Standard Condition** option and click the **Add** button to add a new test condition for which each configuration can provide a unique value.
    c. Select the **Production** panel and enter `Production` for the value of the TestFlowId test condition in the table.
    d. Click **OK** to close the Test Program Editor.
6. Complete the following steps to obtain the value of the TestFlowId test condition in the test program.
    a. Select the `ProcessSetup` sequence.
    b. Add a Get Test Information step after the Open Sessions step.
    c. On the Step Settings pane, click the **Get Test Information** tab.
    d. Enter `TestFlowId` for the **Setting Name** in the first row of the table.
    e. Set the Destination Expression to `Locals.Flow`.
    f. Highlight and right-click `Locals.Flow` and select **Create "Locals.Flow"»String** from the context menu to create a new local variable.
    g. Add a Message Popup step after the Get Test Information step in the Setup step group.
    h. On the Step Settings pane, click the **Text and Buttons** tab.
    i. Set the Message Expression to `"Starting test flow: " + Locals.Flow`.
7. Complete the following steps to specify a code module for the Continuity Test step.
    a. Select the `MainSequence` sequence and the Continuity Test step.
    b. On the Step Settings pane, click the **Module** tab.

c. Click the **Browse for Assembly** button located to the right of the **Assembly** control.

d. Browse to `<TestStand Public>\Tutorial\NI_SemiconductorModule\Basic Test Program\DotNET\Code Modules\BasicTestProgramTutorial.dll` and click **Open**. Select the **Use a relative path for the file you selected** option when prompted and click **OK**.

e. In the **Root Class** control, select the `BasicTestProgramCodeModule` class.

f. In the **.NET Invocation** control, select the `TestContinuity` method.

g. In the Parameter Table, configure the following parameter values:

Parameter	Value
semiconductorModuleContext	`Step.SemiconductorModuleContext`
pinsAndPinGroups	`{"AllPins"}`

8. Save the sequence file.
9. Complete the following steps to configure the Built-in Simulated Handler Driver.

   a. Select **Semiconductor Module»Configure Station** or click the **Configure Station** button on the TSM toolbar to launch the Configure Station Settings dialog box.

   b. On the General tab of the Configure Station Settings dialog box, place a checkmark in the **Enable Handler/Prober Driver (Real or Simulated)** checkbox.

   c. Select **Built-in Simulated Handler Driver** from the **Handler/Prober Driver** drop-down menu.

   d. Click the **Configure Handler/Prober Driver** button to launch the Configure Built-in Simulated Handler dialog box.

e. Set the **Number of DUTs to Test** option to `10` to specify how many DUTs the simulated lot contains and click **OK** to close the Configure Built-in Simulated Handler dialog box.

f. Select the Advanced tab in the Configure Station Settings dialog box and click the **Result Processing** button to launch the Result Processing dialog box.

g. Enable the **Debug Test Results Log** result processing plug-in.

h. Disable the TestStand **Report** result processing plug-in.

i. Enable the Display option for the **Lot Summary Report** result processing plug-in.

j. Click **OK** to close the Result Processing dialog box, and click **OK** to close the Configure Station Settings dialog box.

10. Complete the following steps to configure a lot to run.

    a. Select **Semiconductor Module»Configure Lot** or click the **Configure Lot** button on the TSM toolbar to launch the Configure Lot Settings dialog box.

    b. Select **Production** in the **Test Program Configuration** option.

    c. Click **OK** to close the Configure Lot Settings dialog box.

Note You can also use the **Active Configuration** drop-down menu on the TSM toolbar to set the configuration. TSM uses previously set values for the other lot settings that appear in the Configure Lot Settings dialog box.

11. Click the **Show Lot Statistics Viewer** button on the TSM toolbar to launch a floating window that shows the binning results of the test program execution. The execution control buttons, such as **Start/Resume Lot** and **End Lot** and the **Configuration** drop-down menu are also available in the Lot Statistics Viewer for convenience.

12. Click the **Start Lot** button on the TSM toolbar or in the Lot Statistics Viewer to execute the test program. The test program launches a dialog box that

shows the value of the TestFlowId test condition. Click **OK** to close the dialog box. TestStand generates and displays the Lot Summary Report on the Report pane of the Execution window. Review the results of the lot.

13. Click the **Active Report** button to switch to the Debug Test Results Log and review the results of each test.

Note The Production configuration overrides the test limits the sequence specifies. The `ProductionLimits.txt` test limits file provides the test limits values to use for each test instead of the values entered in the Semiconductor Multi Test step.

14. Close the Execution and Sequence File windows.

See Also

Getting Started with TSM

Glossary

Tutorial: Importing Test Limits from a File

Tutorial: Importing Test Limits from a File (TSM)

Complete the following steps to open an existing sequence file and create a test limits file for importing modified test limits.

1. Open `<TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\Importing Limits.seq`. This sequence file contains a sequence with Semiconductor Multi Test steps for continuity, leakage, and functional tests. Each step has been configured with test limits. Select each step and review the tests configured on the Tests tab in the Step Settings pane.
2. Complete the following steps to export the current test limits from the sequence file to a test limits file.

a. Select **Semiconductor Module»Export Test Limits from Importing Limits.seq** or click the **Export Test Limits from Importing Limits.seq** button on the TSM toolbar.

b. Save the file as `<TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\ProductionLimits.txt`.

c. Click **OK** in the Export Complete dialog box to acknowledge that the export succeeded.

3. Complete the following steps to create a new test limits file with stricter limits.

a. Open `<TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\ProductionLimits.txt` in a text editor or in Microsoft Excel.

b. Locate the columns for LowLimitExpression and HighLimitExpression and make the changes to each test as shown in the following table.

Note The test limits file uses a tab-delimited file format, and columns might not be aligned in some text editors. Count the number of tabs in a row to match the values in a column to the column header.

Step Name	LowLimit Expression Value	HighLimit Expression Value
Continuity Test	6	9
Leakage Test	35	90

c. Save the file as `<TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\QALimits.txt`.

4. Complete the following steps to import limits from the `QALimits.txt` test limits file.

a. In the TestStand Sequence Editor, select **Semiconductor Module»Import Test Limits into Importing Limits.seq** or click the **Import Test Limits into Importing Limits.seq** button on the TSM toolbar.

b. Select the <TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\QALimits.txt file and click **Open**.

c. Click **OK** in the Import Test Limits into Sequence File dialog box.

d. Click **OK** in the Import Complete dialog box.

e. Select the Continuity and Leakage Test steps and review the test limits on the Tests tab of the Step Settings pane. The values in the Low Limit and High Limit columns are now updated to the values from the QALimits.txt test limits file.

5. Complete the following steps to remove information from the QALimits.txt test limits file that the Import/Export Test Limits tool does not update to simplify the test limits file.

   a. Open <TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\QALimits.txt in a text editor or in Microsoft Excel.

   b. Delete the <SequenceName>, <StepName>, <StepId>, and <TestName> columns because the <TestNumber> column uniquely identifies each test.

   c. Delete the <Pin>, <ComparisonType>, <ScalingFactor>, <Units>, <EvalutationType>, and <FailBin> columns because you will not be modifying those values in this tutorial.

   d. Save the QALimits.txt file and import the test limits using the same procedure as in Step 4.

   e. Select the Continuity and Leakage Test steps and review the test limits on the Tests tab of the Step Settings pane. The limits remain the same even though the Import/Export Test Limits tool removed columns from the test limits file.

   f. Click the **Undo** button or press <Ctrl-Z> to undo the changes that importing the limits file made.

6. Complete the following steps to update only specific test steps.

a. Open <TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\QALimits.txt in a text editor or in Microsoft Excel.

b. Delete the rows in the limits file for the Leakage tests that use 0V. These tests have test numbers 201, 203, 205, and 207.

c. Save the QALimits.txt file and import the test limits using the same procedure as in Step 4.

   In Step 4.c, the Import Tests Limits into Sequence File dialog box enables the Update limits in matching tests option by default. This setting specifies to import only the limits to tests that have matching rows in the test limits file. A row matches a test if the row and test use the same identifier columns: <SequenceName>, <StepName>, <StepId>, <TestName>, and <TestNumber>. If any of the identifier columns are missing, the row might match multiple tests in the sequence file.

   Note The Import Complete dialog box warns you that some of the tests in the sequence file were not found in the test limits file. You can ignore this warning because you intentionally removed those tests from the test limits file so that the Import/Export Test Limits tool would not update them.

d. Select the Leakage Test step and review the test limits on the Tests tab of the Step Settings pane. The values of the Low and High Limits are now updated to the values from the QALimits.txt test limits file for the 5V leakage tests but not for the 0V leakage tests.

e. Click the **Undo** button or press <Ctrl-Z> to undo the changes that importing the limits file made.

7. Complete the following steps to delete the existing tests in the sequence file and replace them with the tests from the test limits file.

a. Open <TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\ProductionLimits.txt in a text editor or in Microsoft Excel.

b. This tutorial uses the Replace all tests in matching steps import mode. Because this mode requires more information in the test limits file to identify which step the tests limits belong to, delete only the <StepId> column. Do not delete any other columns because the Replace all tests in matching steps mode leaves values for test settings columns as the default value if the test limits file does not specify a value.

c. Delete the rows in the test limits file for tests 101, 102, 103, and 104.

d. Update the values for the <LowLimitExpression> and <HighLimitExpression> columns for test 100 to 6 and 9, respectively.

e. Save the changes to the test limits file.

f. Select **Semiconductor Module»Import Test Limits into Importing Limits.seq** or click the **Import Test Limits into Importing Limits.seq** button on the TSM toolbar to launch the Import Test Limits dialog box.

g. Select the <TestStand Public>\Tutorial\NI_SemiconductorModule\Importing Test Limits\LabVIEW\ProductionLimits.txt file and click **Open**.

h. In the Import Test Limits into Sequence File dialog box, enable the **Replace all tests in matching steps** option.

i. Click **OK** in the Import Test Limits into Sequence File dialog box.

j. Click **OK** in the Import Complete dialog box.

k. Select the Continuity Test and review the test limits on Tests tab of the Step Settings pane. The tests are now updated to match the tests from the limits file, and the Continuity Test contains only test 100.

l. Click the **Undo** button or press <Ctrl-Z> to undo the changes that importing the limits file made.

8. Complete the following steps to create a test program configuration that loads the limits file when the test program begins execution.

a. In the TestStand Sequence Editor, select **Semiconductor Module»Edit Test Program: Importing Limits.seq** or click the **Edit Test Program: Importing Limits.seq** button on the TSM toolbar.
b. On the Test Limits Files panel, click the **Add Test Limits File** button to add a new file reference to the test program.
c. Enter `Production Limits` as the name to identify the new limits file in the test program.
d. Enter `ProductionLimits.txt` in the **Test Limits File Path** control. The filename you enter is a relative path from the sequence file to the test limits file, `ProductionLimits.txt`, which is located in the same directory as the sequence file.
e. Select the **Production** panel to edit the Production configuration. Select **Production Limits** in the **Test Limits File** control to configure the Production configuration to load the test limits file when the lot begins execution.
f. Click **OK** to close the Test Program Editor.
g. Select **Configure»Result Processing** to launch the Result Processing dialog box.
h. Enable the **Debug Test Results Log** result processing plug-in and select it as the Display report. Disable all other report types.
i. Click **OK** to close the Result Processing dialog box.
j. Select **Semiconductor Module»Configure Lot** or click the **Configure Lot** button on the TSM toolbar to launch the Configure Lot Settings dialog box.
k. Select **Production** for the **Test Program Configuration**.
l. Click **OK** to exit the Configure Lot Settings dialog box.
m. Click the **Start Lot** button on the TSM toolbar to execute the test program. TestStand generates and displays a report on the Report pane of the Execution window.
n. Review the Debug Test Results Log and review the limits of each test. The Production configuration overrides the test limits the sequence

specifies. The `ProductionLimits.txt` test limits file provides the test limits values to use for each test instead of the values entered in the Semiconductor Multi Test step.

9. Complete the following steps to import the same tests to multiple steps.
    a. Add an If step below the Get Current Level step in the `MainSequence` sequence.
    b. Set the Conditional Expression of the If step to `Locals.LowCurrent`, a local variable that stores the value of the LowCurrent test condition obtained from the current configuration in the Get Current Level step. This value is `True` for the Production configuration.
    c. Add an Else step between the If and End steps.
    d. Move the Continuity Test step between the If and Else steps.
    e. Create a copy of the Continuity Test step and place it between the Else and End steps. Rename the original version of the Continuity Test step between the If and Else steps to `Low Current Continuity Test`.
    f. In the Low Current Continuity Test step, set the Low Limits for test number 101, 102, 103, and 104 to `2`.
    g. Click the **Start Lot** button on the TSM toolbar to execute the test program. TestStand generates and displays a report on the Report pane of the Execution window.
    h. Review the Debug Test Results Log and review the limits of each test. The Production configuration is still configured to override the test limits the sequence specifies. The `ProductionLimits.txt` test limits file provides the test limits values for test 100 of the Continuity Test step but not for tests 102, 103, and 104. Because the test limits file designates only a `<TestNumber>` and does not designate a `<StepId>`, the test limits file overrides the test limits in every step that contains test 100.

See Also

Explore a basic semiconductor test program

Glossary

Test Program Overview

TSM Example Programs

Use the TSM example programs, located in the <TestStand Public>\Examples\NI_SemiconductorModule directory, to learn more about how to solve specific issues in a semiconductor test program.

Accelerometer (TSM)

The <TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer directory contains the following examples:

- Accelerometer with LabVIEW
- Accelerometer with .NET

Note The NI TestStand 2016 Semiconductor Module and later natively support digital pattern instruments that use the NI-Digital Pattern Driver and legacy digital waveform instruments that use the NI-HSDIO driver, such as the PXIe-6556. Use the TSM Code Module API that corresponds to the type of digital instrument the test system includes.

Accelerometer with LabVIEW (TSM)

Purpose

This example demonstrates several features of TSM in a test program that makes common measurements with the NI-Digital Pattern driver to test an imagined accelerometer part. You can use this example as a starting point for your test programs.

Example File Locations

<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\Accelerometer.seq

Highlighted Features

- Pin map
- Multisite
- Custom instruments
- Binning
- Specifications files
- Limits files
- Test program configurations
- Virtual pins
- Offline Mode

Major API

TSM Code Module API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- You must have NI-Digital Pattern Driver 19.0 or later installed, and you must have two NI-Digital Pattern instruments named `HSD_6570_C1_S02` and `HSD_6570_C1_S04`, respectively, as defined in Measurement & Automation Explorer (MAX).
- You must have NI-DCPower 20.6 or later installed, and you must have two NI-DCPower instruments named `SMU_4143_C1_S06` and `SMU_4143_C1_S07`, respectively, as defined in MAX.
- You must have NI-SCOPE 15.0 or later installed, and you must have an NI-SCOPE instrument named `SCOPE_5105_C1_S08` as defined in MAX.
- You must have NI-SWITCH 17.0 or later installed, and you must have a PXI-2567 relay driver module named `RELAY_2567_C1_S09`, as defined in MAX.
- (Offline Mode) You must meet the requirements to run the test program in Offline Mode.

- This example uses the Batch process model.

Notes

- You can view the test program in the TestStand Sequence Editor and partial code modules in LabVIEW without the required NI instrument drivers installed. Install the drivers to view the full code modules in LabVIEW. Visit `ni.com/info` and enter the Info Code `rddrau` to access the latest software drivers and updates.
- (Offline Mode) The NI-Digital Pattern, NI-DCPower, and NI-SCOPE instruments and the PXI-2567 relay driver module are not required to run this example in Offline Mode, but you must install the required instrument drivers.

How to Use This Example

Complete the steps in the following sections to learn about the test program components. You can also use this example in offline mode.

Pin Map

Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to open the Accelerometer pin map file in the Pin Map Editor. The pin map file defines the following information:

- Two NI-Digital Pattern instruments named `HSD_6570_C1_S02` and `HSD_6570_C1_S04`. Both instruments belong to the same group so that code modules can access all digital pins on the tester using a single instrument session.
- Two NI-DCPower instruments named `SMU_4143_C1_S06` and `SMU_4143_C1_S07`.
- One NI-SCOPE instrument named `SCOPE_5105_C1_S08`.
- One PXI-2567 relay driver module named `RELAY_2567_C1_S09`.

- Ten DUT pins named `Vcc`, `Gnd`, `SCLK`, `MOSI`, `MISO`, `CS`, `RST`, `MODE`, `Vref_DIO`, and `Vref_OScope`. The `Vref_DIO` and `Vref_OScope` pins are virtual pins that refer to a single `Vref` DUT pin and are used to connect the pin to two different types of instruments, NI-Digital Pattern and NI-SCOPE.
- One relay named `SCOPE_ENABLE_RELAY` per site. The test program uses the `SCOPE_ENABLE_RELAY` relay to control a physical relay that connects the `Vref` DUT pin to the NI-Digital Pattern instrument or to the NI-SCOPE instrument.
- One relay named `NOISE_ENABLE_RELAY` per site. The test program uses the `NOISE_ENABLE_RELAY` relay to control a physical relay that connects the `Vref` DUT pin to a noise source, rather than to the NI-Digital Pattern or NI-SCOPE instruments.
- Three pin groups named `SPI_Port`, `Digital`, and `AllDUTPins`.
- One system relay named `POWER_RELAY`. The test program uses the `POWER_RELAY` relay to control a physical relay that controls a power source.
- Four sites on the tester.
- A series of connections for each site, in which each connection specifies a DUT pin, a site number, an instrument, and an instrument channel.
- Site relay connections that specify to which control line of a relay driver module the `SCOPE_ENABLE_RELAY` relay is connected for a given site.
- Site relay connections that specify to which control line of a relay driver module the `NOISE_ENABLE_RELAY` relay is connected for a given site.
- A system relay connection that specifies whether the power source connected to the `POWER_RELAY` relay is enabled.

Test Program Configurations

Complete the following steps to review the Test Program Configurations that this test program uses.

1. Select **Semiconductor Module»Edit Test Program: Accelerometer.seq** or click the **Edit Test Program: Accelerometer** button on the TSM toolbar.

2. Select the **Configuration Definition** panel. This test program specifies two test conditions that each test program configuration must define:
   - **TestFlowId**—Defines an identifying name for the test flow.
   - **TestTemperature**—Defines the temperature at which to perform the tests.
3. Select each of the individual **Configuration** panels to review the values each test program configuration gives to the specified test conditions.

Bin Definitions

Use the TestStand Sequence Editor to review the bin definitions file associated with the test program.

Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar. The bin definitions file defines software bins that the test program uses and the hardware bins associated with the software bins.

MainSequence, ProcessSetup, and ProcessCleanup Sequences

Complete the following steps to review the `MainSequence`, `ProcessSetup`, and `ProcessCleanup` sequences that this test program uses.

1. On the Sequences pane, select the `MainSequence` sequence and review the objectives each step performs and optionally review the LabVIEW code associated with each step:
   - In the Setup section, if the current test program configuration uses the `Hot` test temperature setting, the test program waits until the temperature controller reaches the specified temperature.
   - To prepare for digital tests, the test program configures the relay for the `Vref` pin to the Digital Pattern instrument.
   - The test program tests continuity, leakage, and idle power consumption on all digital pins.
   - The test program resets the DUT in preparation for SPI port communication.

- The test program enables test mode on the DUT using the SPI port.
- The test program checks the part number of the DUT by reading a register through the SPI port.
- If the current TestFlowId is set to `Quality` the test program checks the part number at different Vcc levels.
- To prepare for analog tests, the test program configures the relay for the `Vref` pin to the NI-SCOPE instrument.
- The test program checks the minimum, maximum, and RMS voltage value on the Vref and uses the value to trim each DUT.
- The test program sets and verifies the Vref register on the DUT based on the previous Vref measurement.
- In the Cleanup section, the test program turns off all instrument output to the DUT in preparation for physical binning by the handler.

2. On the Sequences pane, select the `ProcessSetup` sequence. TestStand calls this sequence once before starting testing. The steps in this sequence initialize the instruments and store the instrument sessions in the SemiconductorModuleContext. There are other steps in this sequence to configure a temperature controller, and to toggle the power source.
3. On the Sequences pane, select the `ProcessCleanup` sequence. TestStand calls this sequence once after testing completes. The steps in this sequence close and reset the instruments.

Specifications, Time Sets, Pin Levels, Patterns, and Waveforms

Select **Semiconductor Module»Launch Digital Pattern Editor** or click the **Launch Digital Pattern Editor** button on the TSM toolbar to open the Digital Pattern Editor. Open the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\Accelerometer.digiproj` digital pattern project file in the Digital Pattern Editor. Use the Digital Pattern Editor to review the following files the Accelerometer test program uses:

- `Accelerometer.specs` — Defines a set of variables and associated numeric values that you can reference in pin levels, time sets, other specifications files, and Shmoo operations.

- `Accelerometer.digitiming` — Defines configuration components of the time sets, including the format and edge placement that shape the digital waveform on a per-pin basis.
- `Accelerometer.digilevels` — Defines voltage levels for digital pins and pin groups connected to a Digital Pattern Instrument and for pins and pin groups connected to an NI-DCPower instrument.
- `SPI - Read Part Number.digipat` — Pattern that reads the part number register from the DUT.
- `SPI - Set Test Mode.digipat` — Pattern that sets the test mode by setting a register on the DUT.
- `SPI - Set Vref Value.digipat` — Pattern that sets the Vref register on the DUT using a source waveform and reads it back using a capture waveform.
- `Set Vref Value Waveform.tdms` — Source waveform configuration used to set the Vref register value.
- `Get Vref Value Waveform.digicapture` — Capture waveform configuration used to obtain the Vref register value.

External Limits Files

Use a text editor or spreadsheet software to open and review the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\Limits\Production Limits.txt` file, which is the tests limits file for the Production configuration of this test program. The test limits file is loaded at run time based on the current test configuration. This file specifies the values to use to evaluate whether a measurement passes or fails. The test program stores the limits loaded from the file with the results.

Running the Test Program

You must meet all the prerequisites to run the test program. To run the test program, click the **Start/Resume Lot** button on the TSM toolbar.

Running the Test Program in Offline Mode

You must install the required instrument drivers and meet all the prerequisites to run the test program on a computer without access to NI instruments. To simulate the instruments the test program needs, click the **Enable Offline Mode** button on the TSM toolbar. To run the test program, click the **Start/Resume Lot** button on the TSM toolbar.

Launch the Test Program Editor and select the Offline Mode panel to view the path to the Offline Mode system configuration file TSM uses to create simulated instruments for `Accelerometer.seq`.

Click the **Disable Offline Mode** button to return to the default TSM behavior.

Accelerometer with .NET (TSM)

Purpose

This example demonstrates several features of TSM in a test program that makes common measurements with the NI-Digital Pattern driver to test an imagined accelerometer part. You can use this example as a starting point for your test programs.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\DotNET\Accelerometer.seq`

Highlighted Features

- Pin map
- Multisite
- Custom instruments
- Binning
- Specifications files
- Limits files
- Test program configurations

- Virtual pins
- Offline Mode

Major API

TSM Code Module API

Prerequisites

- You must have NI-Digital Pattern Driver 19.0 or later installed, and you must have two NI-Digital Pattern instruments named `HSD_6570_C1_S02` and `HSD_6570_C1_S04`, respectively, as defined in Measurement & Automation Explorer (MAX).
- You must have NI-DCPower 20.6 or later installed, and you must have two NI-DCPower instruments named `SMU_4143_C1_S06` and `SMU_4143_C1_S07`, respectively, as defined in MAX.
- You must have NI-SCOPE 15.0 or later installed, and you must have an NI-SCOPE instrument named `SCOPE_5105_C1_S08` as defined in MAX.
- You must have NI-SWITCH 17.0 or later installed, and you must have a PXI-2567 relay driver module named `RELAY_2567_C1_S09`, as defined in MAX.
- (Offline Mode) You must meet the requirements to run the test program in Offline Mode.
- You must have .NET 4.0 support for the NI-DCPower .NET Class Libraries 1.1 or later installed.
- You must have .NET 4.0 support for the NI-SCOPE .NET Class Libraries 2.0 or later installed.
- You must have .NET 4.0 support for the NI-SWITCH .NET Class Libraries 1.1 or later installed.
- This example uses the Batch process model.

Notes

- You can view the test program in the TestStand Sequence Editor and code modules in a C# source code editor

without the required NI instrument drivers installed. Visit `ni.com/info` and enter the Info Code `NETAPIdriversupport` for information about the available NI .NET APIs and the versions of the NI drivers each supports.

- (Offline Mode) The NI-Digital Pattern, NI-DCPower, and NI-SCOPE instruments and the PXI-2567 relay driver module are not required to run this example in Offline Mode, but you must install the required instrument drivers.

How to Use This Example

Complete the steps in the following sections to learn about the test program components.

Pin Map

Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to open the Accelerometer pin map file in the Pin Map Editor. The pin map file defines the following information:

- Two NI-Digital Pattern instruments named `HSD_6570_C1_S02` and `HSD_6570_C1_S04`. Both instruments belong to the same group so that code modules can access all digital pins on the tester using a single instrument session.
- Two NI-DCPower instruments named `SMU_4143_C1_S06` and `SMU_4143_C1_S07`.
- One NI-SCOPE instrument named `SCOPE_5105_C1_S08`.
- One PXI-2567 relay driver module named `RELAY_2567_C1_S09`.
- Ten DUT pins named `Vcc`, `Gnd`, `SCLK`, `MOSI`, `MISO`, `CS`, `RST`, `MODE`, `Vref_DIO`, and `Vref_OScope`. The `Vref_DIO` and `Vref_OScope` pins are virtual pins that refer to a single `Vref` DUT pin and are used to connect the pin to two different types of instruments, NI-Digital Pattern and NI-SCOPE.

- One relay named `SCOPE_ENABLE_RELAY` per site. The test program uses the `SCOPE_ENABLE_RELAY` relay to control a physical relay that connects the `Vref` DUT pin to the NI-Digital Pattern instrument or to the NI-SCOPE instrument.
- One relay named `NOISE_ENABLE_RELAY` per site. The test program uses the `NOISE_ENABLE_RELAY` relay to control a physical relay that connects the `Vref` DUT pin to a noise source, rather than to the NI-Digital Pattern or NI-SCOPE instruments.
- Three pin groups named `SPI_Port`, `Digital`, and `AllDUTPins`.
- One system relay named `POWER_RELAY`. The test program uses the `POWER_RELAY` relay to control a physical relay that controls a power source.
- Four sites on the tester.
- A series of connections for each site, in which each connection specifies a DUT pin, a site number, an instrument, and an instrument channel.
- Site relay connections that specify to which control line of a relay driver module the `SCOPE_ENABLE_RELAY` relay is connected for a given site.
- Site relay connections that specify to which control line of a relay driver module the `NOISE_ENABLE_RELAY` relay is connected for a given site.
- A system relay connection that specifies whether the power source connected to the `POWER_RELAY` relay is enabled.

Test Program Configurations

Complete the following steps to review the Test Program Configurations that this test program uses.

1. Select **Semiconductor Module»Edit Test Program: Accelerometer.seq** or click the **Edit Test Program: Accelerometer** button on the TSM toolbar.
2. Select the **Configuration Definition** panel. This test program specifies two test conditions that each test program configuration must define:
   - **TestFlowId**—Defines an identifying name for the test flow.

- **TestTemperature**—Defines the temperature at which to perform the tests.

3. Select each of the individual **Configuration** panels to review the values each test program configuration gives to the specified test conditions.

Bin Definitions

Use the TestStand Sequence Editor to review the bin definitions file associated with the test program.

Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar. The bin definitions file defines software bins that the test program uses and the hardware bins associated with the software bins.

MainSequence, ProcessSetup, and ProcessCleanup Sequences

Complete the following steps to review the `MainSequence`, `ProcessSetup`, and `ProcessCleanup` sequences that this test program uses.

1. On the Sequences pane, select the `MainSequence` sequence and review the objectives each step performs and optionally review the LabVIEW code associated with each step:
   - In the Setup section, if the current test program configuration uses the `Hot` test temperature setting, the test program waits until the temperature controller reaches the specified temperature.
   - To prepare for digital tests, the test program configures the relay for the `Vref` pin to the Digital Pattern instrument.
   - The test program tests continuity, leakage, and idle power consumption on all digital pins.
   - The test program resets the DUT in preparation for SPI port communication.
   - The test program enables test mode on the DUT using the SPI port.
   - The test program checks the part number of the DUT by reading a register through the SPI port.

- If the current TestFlowId is set to `Quality` the test program checks the part number at different Vcc levels.
- To prepare for analog tests, the test program configures the relay for the `V ref` pin to the NI-SCOPE instrument.
- The test program checks the minimum, maximum, and RMS voltage value on the Vref and uses the value to trim each DUT.
- The test program sets and verifies the Vref register on the DUT based on the previous Vref measurement.
- In the Cleanup section, the test program turns off all instrument output to the DUT in preparation for physical binning by the handler.

2. On the Sequences pane, select the `ProcessSetup` sequence. TestStand calls this sequence once before starting testing. The steps in this sequence initialize the instruments and store the instrument sessions in the SemiconductorModuleContext. There are other steps in this sequence to configure a temperature controller, and to toggle the power source.
3. On the Sequences pane, select the `ProcessCleanup` sequence. TestStand calls this sequence once after testing completes. The steps in this sequence close and reset the instruments.

Specifications, Time Sets, Pin Levels, Patterns, and Waveforms

Select **Semiconductor Module»Launch Digital Pattern Editor** or click the **Launch Digital Pattern Editor** button on the TSM toolbar to open the Digital Pattern Editor. Open the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\DotNET\Accelerometer.digiproj` digital pattern project file in the Digital Pattern Editor. Use the Digital Pattern Editor to review the following files the Accelerometer test program uses:

- `Accelerometer.specs` — Defines a set of variables and associated numeric values that you can reference in pin levels, time sets, other specifications files, and Shmoo operations.
- `Accelerometer.digitiming` — Defines configuration components of the time sets, including the format and edge placement that shape the digital waveform on a per-pin basis.

- `Accelerometer.digilevels` — Defines voltage levels for digital pins and pin groups connected to a Digital Pattern Instrument and for pins and pin groups connected to an NI-DCPower instrument.
- `SPI - Read Part Number.digipat` — Pattern that reads the part number register from the DUT.
- `SPI - Set Test Mode.digipat` — Pattern that sets the test mode by setting a register on the DUT.
- `SPI - Set Vref Value.digipat` — Pattern that sets the Vref register on the DUT using a source waveform and reads it back using a capture waveform.
- `Set Vref Value Waveform.tdms` — Source waveform configuration used to set the Vref register value.
- `Get Vref Value Waveform.digicapture` — Capture waveform configuration used to obtain the Vref register value.

External Limits Files

Use a text editor or spreadsheet software to open and review the `<TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\DotNET\Limits\Production Limits.txt` file, which is the tests limits file for the Production configuration of this test program. The test limits file is loaded at run time based on the current test configuration. This file specifies the values to use to evaluate whether a measurement passes or fails. The test program stores the limits loaded from the file with the results.

Running the Test Program

You must meet all the prerequisites to run the test program. To run the test program, click the **Start/Resume Lot** button on the TSM toolbar.

Running the Test Program in Offline Mode

You must install the required instrument drivers and meet all the prerequisites to run the test program on a computer without access to NI instruments. To simulate the instruments the test program needs, click the **Enable Offline Mode** button

on the TSM toolbar. To run the test program, click the **Start/Resume Lot** button on the TSM toolbar.

Launch the Test Program Editor and select the Offline Mode panel to view the path to the Offline Mode system configuration file TSM uses to create simulated instruments for `Accelerometer.seq`.

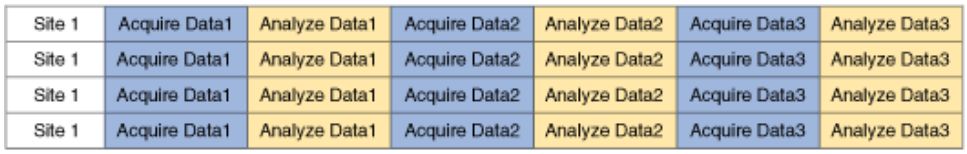
Click the **Disable Offline Mode** button to return to the default TSM behavior.

Asynchronous Analysis (TSM)

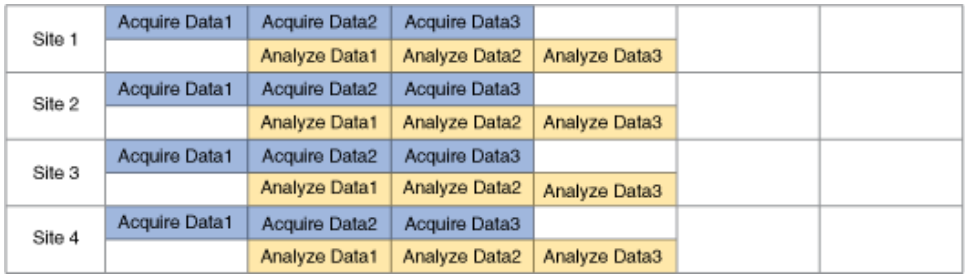
Purpose

This example demonstrates a test program that must perform lengthy analysis on data acquired from an instrument. The test program performs the analysis in an asynchronous thread. The following two figures demonstrate how performing the analysis asynchronously to the data acquisition reduces the overall duration of the test program.

The following figure shows a test program thread usage without asynchronous analysis.



The following figure shows a test program thread usage with asynchronous analysis.



Example File Locations

`<TestStand Public>\Examples\NI_SemiconductorModule\Test Optimizations\Asynchronous Analysis\LabVIEW\Asynchronous Analysis.seq`

Highlighted Features

- Multisite
- Asynchronous code modules

Major API

None

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

1. Complete the following steps to review the pin map file associated with the test program.
    a. Select **Semiconductor Module»Edit Pin Map File**, click the **Edit Pin Map File** button on the TSM toolbar, or launch the Test Program Editor and click the **Open file for edit** button located to the right of the Pin Map File Path display on the Pin Map panel to launch the Pin Map Editor. Open the `<TestStand Public>\Examples\NI_SemiconductorModule\Test Optimizations\Asynchronous Analysis\LabVIEW\Asynchronous Analysis.pinmap` file in the Pin Map Editor.

    The pin map file defines the following information:

    - One DUT pin named `A`.
    - One custom instrument named `Dev1` with an `instrumentTypeId` attribute value of `CustomInstrument` and one channel group that contains four channels.
    - Four sites on the tester.

- A series of connections for each site, in which each connection specifies DUT pin `A`, a site number, the custom instrument `Dev1`, and an instrument channel.

2. Open `<TestStand Public>\Examples\NI_SemiconductorModule\Test Optimizations\Asynchronous Analysis\LabVIEW\Asynchronous Analysis.seq` and review the steps in the sequence.
   - Each Acquire Test Data step is a Semiconductor Multi Test step that runs a LabVIEW code module that acquires data for each site and exports a reference to the data to a local variable. Because the pin map specifies a channel for each site connected to pin `A`, the test runs one instance of the code module that acquires data for all four sites.
   - Each Analyze Test Data step runs a VI asynchronously to perform analysis on the data acquired in the preceding Acquire Test Data step. TestStand passes to the VI the local variable that contains the reference to the test data and stores the analysis result the VI returns in another local variable.
   - Each Wait for Analysis of Test Data step waits for the asynchronous thread running the analysis VI to complete.
   - Each Evaluate Analysis Result step is a Semiconductor Multi Test step with no code module. The step evaluates the value of the local variable that contains the analysis result against a set of limits. The Tests tab in the Step Settings pane shows the limits. If the value of the analysis result is within the limits, the test passes. Otherwise, the test fails.
3. On the Sequences pane, select the `ProcessSetup` sequence. The Launch Resource Usage Profile step launches the Execution Profiler to demonstrate how the analysis threads run in parallel to the data acquisition threads. The Open and Store Instrument Sessions step in this sequence initializes the custom instrument.
4. On the Sequences pane, select the `ProcessCleanup` sequence. The Close Instrument Sessions step in this sequence closes the custom instrument session.
5. Open `<TestStand Public>\Examples\NI_SemiconductorModule\Test Optimization\Asynchronous Analysis\LabVIEW\Async`

hronous Analysis.lvproj to review the code modules the test program uses.

6. Complete the following steps to execute the test program.
    a. Select **Configure»Model Options** to launch the Model Options dialog box.
    b. Set the **Number of Test Sockets** to 4 and click **OK** to close the Model Options dialog box.
    c. Click the **Start Lot** button on the TSM toolbar to run the test program.
    d. The Execution Profiler automatically launches when execution begins. The profiler shows each thread running in the test program and demonstrates that the analysis steps are running in parallel with the acquisition steps.

Custom Instruments (TSM)

TSM provides built-in support for commonly used instruments, such as NI-DCPower and NI-HSDIO. If you need to use an instrument TSM does not natively support, complete the following steps to define a custom instrument.

1. Based on the instrument driver behavior in a multisite environment, complete the following step:
    - If each channel operates independently and can be accessed in separate threads with different timing, the instrument operates with one session per channel, similar to NI-DCPower instruments. Copy the <TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel\LabVIEW directory to a new location and open the Session per Channel.seq file in the new location.
    - If all channels operate synchronously, such that they must all be accessed together, the instrument operates with one session per instrument, similar to NI-HSDIO instruments. Copy the <TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Instrument\LabVIEW directory to a new location and open the Session per Instrument.seq file in the new location.

- If the instrument has groups of channels that must be accessed synchronously, the instrument operates with one session per channel group. Copy the <TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel Group\LabVIEW directory to a new location and open the Session per Channel Group.seq in the new location.

2. Use the Pin Map Editor to specify instrument channels and IDs.
3. Modify the related VI code modules in the example project to create a communication layer between the pin map and the instrument. Ensure that you update the session data inputs and outputs to a datatype that matches the expected data for the custom instrument.

Session per Channel (TSM)

Purpose

This example demonstrates how to define instruments that TSM does not natively support, how to create API for the instruments, and how to use the API in a code module for a TSM test program. Each instrument in this example has unique session data for each channel, similar to an NI-DCPower instrument. The code module opens one session per channel.

Example File Location

<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel\LabVIEW\Session per Channel.seq

Highlighted Features

- Custom instruments
- Pin map

Major API

TSM Code Module API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

1. Complete the following steps to review the pin map file associated with the test program.
    a. Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to open the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel\LabVIEW\Session per Channel.pinmap` file in the Pin Map Editor.

    The pin map file defines the following information:

    - Two custom instruments named `dev1` and `dev2`, respectively, each with an `instrumentTypeId` attribute value of `SessionPerChannelInst` and eight available channels but no channel group
    - Four DUT pins named `A`, `B`, `C`, and `D`, respectively
    - Four sites on the tester
    - A series of connections for each site, in which each connection specifies a DUT pin, a site number, an instrument, and an instrument channel
2. Complete the following steps to review the code module that communicates with the custom instruments.
    a. Click the LabVIEW Module tab on the Step Settings pane. The Project Path control specifies a file path to the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel\LabVIEW\Session per Channel.lvproj` LabVIEW project file.

b. Click the **Edit LabVIEW Project** button, located to the right of the Project Path control, to open `Session per Channel.lvproj` in LabVIEW.

c. In the Project Explorer window, expand the **Custom Instrument API** folder.

d. Open and review the following VIs that correspond to operations TSM performs on the pin map:

- **Session per Channel - Get All Instrument Names.vi**—Obtains all instruments of the `SessionPerChannelInst` instrument type, as defined in the pin map file, so that the initialization code module can open sessions and store them in the `SemiconductorModuleContext` object.
- **Session per Channel - Get All Sessions.vi**—Obtains all stored sessions so that the clean up code module can close the sessions that were previously opened and stored in the `SemiconductorModuleContext` object.
- **Session per Channel - Pins To Sessions.vi**—Queries the DUT pins and returns the associated instruments and channel lists associated with a `SemiconductorModuleContext` object.
- **Session per Channel - Store Instrument Session.vi**—Stores the sessions in the `SemiconductorModuleContext` object so that other parts of the test program can access the sessions.

3. In TestStand, select the **ProcessSetup** sequence on the Sequences pane. The step in this sequence initializes the instrument sessions.
4. On the Sequences pane, select the **ProcessCleanup** sequence. The step in this sequence closes the instrument sessions.
5. Click the **Start Lot** button on the TSM toolbar to run the test program.

Session per Channel Group (TSM)

Purpose

This example demonstrates how to define instruments that TSM does not natively support, how to create API for the instruments, and how to use the API in a code module for a TSM test program. Each instrument in this example has two unique sessions, each with two channels that share a session. The code module opens one session per channel group.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel Group\LabVIEW\Session per Channel Group.seq`

Highlighted Features

- Custom instruments
- Pin map

Major API

TSM Code Module API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

1. Complete the following steps to review the pin map file associated with the test program.

a. Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to open the <TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel Group\LabVIEW\Session per Channel Group.pinmap file in the Pin Map Editor.

The pin map file defines the following information:

- Two custom instruments named dev1 and dev2, respectively, each with an instrumentTypeId attribute value of SessionPerChannelGroupInst, two channel groups, and four available channels contained in each channel group
- Four DUT pins named A, B, C, and D, respectively
- Four sites on the tester
- A series of connections for each site, in which each connection specifies a DUT pin, a site number, an instrument, and an instrument channel

2. Complete the following steps to review the code module that communicates with the custom instruments.

a. Click the LabVIEW Module tab on the Step Settings pane. The Project Path control specifies a file path to the <TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Channel Group\LabVIEW\Session per Channel Group.lvproj LabVIEW project file.

b. Click the **Edit LabVIEW Project** button, located to the right of the Project Path control, to open Session per Channel Group.lvproj in LabVIEW.

c. In the Project Explorer window, expand the **Session per Channel Group API** folder.

d. Open and review the following VIs that correspond to operations TSM performs on the pin map:

- **Session per Channel Group - Get All Instrument Names.vi**—Obtains all instruments of the SessionPerChannelGroupInst

instrument type, as defined in the pin map file, so that the initialization code module can open sessions and store them in the `SemiconductorModuleContext` object.

- **Session per Channel Group - Get All Sessions.vi**—Obtains all stored sessions so that the clean up code module can close the sessions that were previously opened and stored in the `SemiconductorModuleContext` object.
- **Session per Channel Group - Pins To Sessions.vi**—Queries the DUT pins and returns the associated instruments and channel lists associated with a `SemiconductorModuleContext` object.
- **Session per Channel Group - Store Instrument Session.vi**—Stores the sessions in the `SemiconductorModuleContext` object so that other parts of the test program can access the sessions.

3. In TestStand, select the **ProcessSetup** sequence on the Sequences pane. The step in this sequence initializes the instrument sessions.
4. On the Sequences pane, select the **ProcessCleanup** sequence. The step in this sequence closes the instrument sessions.
5. Click the **Start Lot** button on the TSM toolbar to run the test program.

Session per Instrument (TSM)

Purpose

This example demonstrates how to define instruments that TSM does not natively support, how to create API for the instruments, and how to use the API in a code module for a TSM test program. Each instrument in this example shares session data across the entire instrument, similar to an NI-HSDIO instrument. The code module opens one session per instrument.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Instrument\LabVIEW\Session per Instrument.seq`

Highlighted Features

- Custom instruments
- Pin map

Major API

TSM Code Module API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

1. Complete the following steps to review the pin map file associated with the test program.
    a. Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to open the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Instrument\LabVIEW\Session per Instrument.pinmap` file in the Pin Map Editor.

    The pin map file defines the following information:

    - Two custom instruments named `dev1` and `dev2`, respectively, each with an `instrumentTypeId` attribute value of `SessionPerInstrument` and eight available channels contained in a single channel group
    - Four DUT pins named `A`, `B`, `C`, and `D`, respectively
    - Four sites on the tester

- A series of connections for each site, in which each connection specifies a DUT pin, a site number, an instrument, and an instrument channel

2. Complete the following steps to review the code module that communicates with the custom instruments.
   a. Click the LabVIEW Module tab on the Step Settings pane. The Project Path control specifies a file path to the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments\Session per Instrument\LabVIEW\Session per Instrument.lvproj` LabVIEW project file.
   b. Click the **Edit LabVIEW Project** button, located to the right of the Project Path control, to open `Session per Instrument.lvproj` in LabVIEW.
   c. In the Project Explorer window, expand the **API** folder.
   d. Open and review the following VIs that correspond to operations TSM performs on the pin map:
      - **Session per Instrument - Get All Instrument Names.vi**—Obtains all instruments of the `SessionPerInstrument` instrument type, as defined in the pin map file, so that the initialization code module can open sessions and store them in the `SemiconductorModuleContext` object.
      - **Session per Instrument - Get All Sessions.vi**—Obtains all stored sessions so that the clean up code module can close the sessions that were previously opened and stored in the `SemiconductorModuleContext` object.
      - **Session per Instrument - Pins To Sessions.vi**—Queries the DUT pins and returns the associated instruments and channel lists associated with a `SemiconductorModuleContext` object.
      - **Session per Instrument - Store Instrument Session.vi**—Stores the sessions in the `SemiconductorModuleContext` object so that other parts of the test program can access the sessions.

3. In TestStand, select the **ProcessSetup** sequence on the Sequences pane. The step in this sequence initializes the instrument sessions.
4. On the Sequences pane, select the **ProcessCleanup** sequence. The step in this sequence closes the instrument sessions.
5. Click the **Start Lot** button on the TSM toolbar to run the test program.

Grading (TSM)

Purpose

This example demonstrates how to use the Set and Lock Bin step to grade DUTs based on different test criteria.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Grading\LabVIEW\Grading.seq`

Highlighted Features

- Binning
- Set and Lock Bin step

Major API

None

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

1. Use the TestStand Sequence Editor to complete the following steps to review the bin definitions file associated with the test program.

a. Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar.
b. The bin definitions file defines one Fail bin, one Error bin, and three Pass bins. The Pass bins correspond to the following three different grades: Good, Better, and Best.

2. Review the three Semiconductor Multi Test steps in the `MainSequence` sequence.
   a. The first step takes and stores the measurement in a local variable (`Locals.Measurement`).
   b. Each step compares the measurement against a different set of limits.
   c. Preconditions on the lower grade tests prevent those tests from running if the DUT passed a higher grade test.
   d. The Step Failure Option on the higher grade tests ensure that the DUT does not fail if the higher grade tests fail.
3. Review the If block at the end of the `MainSequence` sequence.
   a. The steps in the block execute only if the DUT passed all tests.
   b. The Set and Lock Bin steps assign a bin to the DUT based on the highest grade test that passed.
4. Click the **Start Lot** button on the TSM toolbar to run the test program.

Multisite Programming Scenarios (TSM)

Purpose

This example demonstrates how to address several multisite use cases.

Example File Locations

`<TestStand Public>\Examples\NI_SemiconductorModule\Multisite Programming Scenarios\LabVIEW\MultisiteScenarios.seq`

Highlighted Features

Testing Multiple Sites in Parallel

Major API

TSM Code Module API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- This example uses the Batch process model.

How to Use This Example

Complete the following steps to use this example.

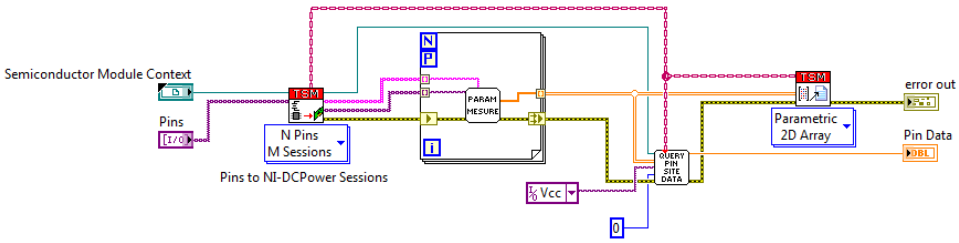
1. Open `<TestStand Public>\Examples\NI_SemiconductorModule\Multisite Programming Scenarios\LabVIEW\MultisiteScenarios.seq`.
2. Review the following steps and corresponding code modules:
   - Simple Parametric Measurement
   - Query Pin/Site Measurement for Unsupported Measurement Type
   - Source-Wait-Measure Parametric Test
   - Source Power Supply Pins and Measure Digital Pins
   - Multiple Measurements on Multiple Pins
   - Source All Digital Pins, Source-Wait-Measure Each Digital Pin
   - Functional Test Using NI-Digital Pattern Driver
   - Functional Test for Multiple Registers Using NI-Digital Pattern Driver
   - Functional Test Using NI-HSDIO Hardware Compare
   - Functional Test for Multiple Registers
   - Instrument Multiplexed Across Sites
   - Instrument Multiplexed Across Sites, Multiple Measurements

See Also

Multisite Programming Techniques

Simple Parametric Measurement (TSM)

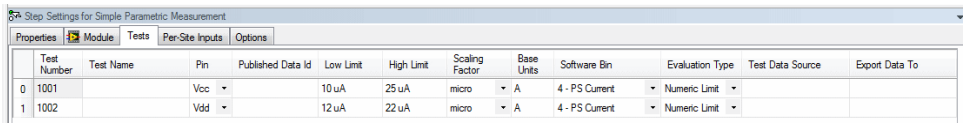
The following figure shows how to take a parametric measurement on one or multiple pins.



This example includes the following main tasks:

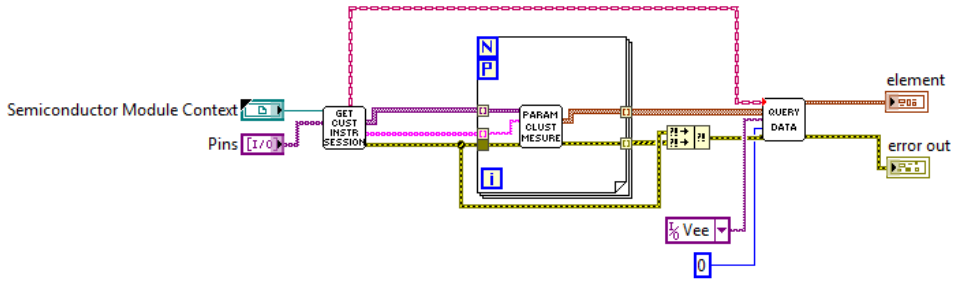
1. Query for the sessions for the pin(s) you want to measure.
2. Use a parallel For Loop to iterate on every instrument you need to call to take the multisite measurement.
3. Use the Pin Query Context with a pin-based instance of the Publish Data VI to publish the measurement.
4. Query for a measurement value for a specific pin and site combination. The source code for the query is in the Query Pin Site Data VI.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each pin, as shown in the following figure:



Query Pin/Site Measurement for Unsupported Measurement Type (TSM)

The following figure shows how to query for measurement data for a pin and site combination if the Extract Pin Data VI does not support the measurement type.

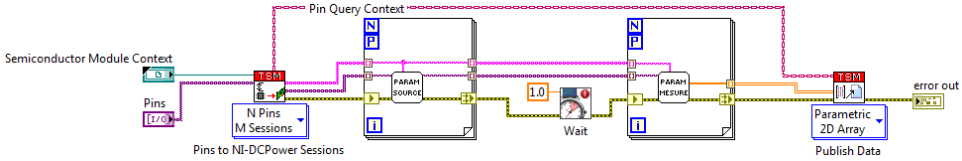


This example includes the following main tasks:

1. Query for the custom instrument sessions for the pin(s) connected to the NI-DCPower instrument.
2. Use a parallel For Loop to iterate on every instrument to create simulated data.
3. Query for measurement data for a specific pin and site combination.

Source-Wait-Measure Parametric Test (TSM)

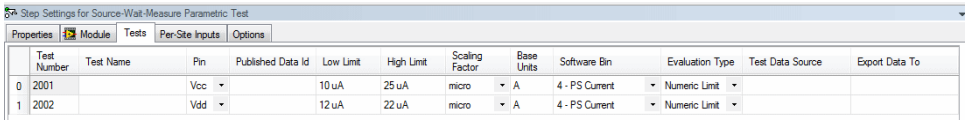
The following figure shows how to source a voltage on the test pin(s), wait for the voltage to settle, and take a measurement.



This example includes the following main tasks:

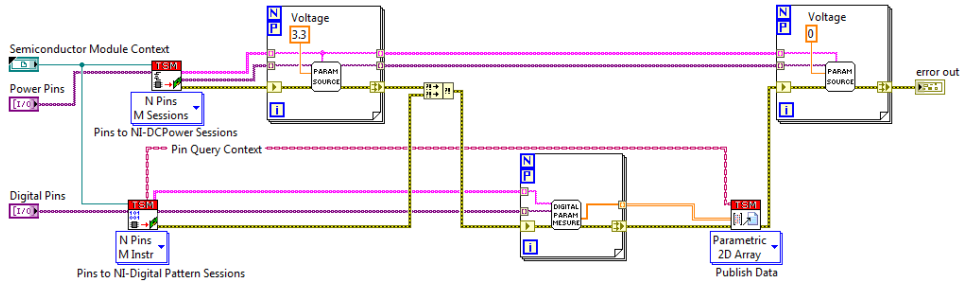
1. Query for the sessions for the pin(s) to source and measure.
2. Use a parallel For Loop to iterate on every instrument you need to call to source a value on the test pin(s).
3. Wait for the voltage to settle on all pins and on all sites.
4. Use a parallel For Loop to iterate on every instrument you need to call to take the multisite measurement.
5. Use the Pin Query Context with a pin-based instance of the Publish Data VI to publish the measurement.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each pin, as shown in the following figure:



Source Power Supply Pins and Measure Digital Pins (TSM)

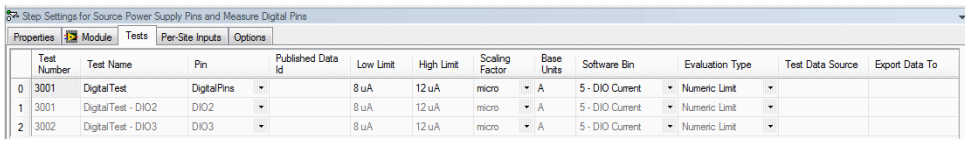
The following figure shows how to source a voltage on the power supply pin(s) and take a measurement on digital I/O pins.



This example includes the following main tasks:

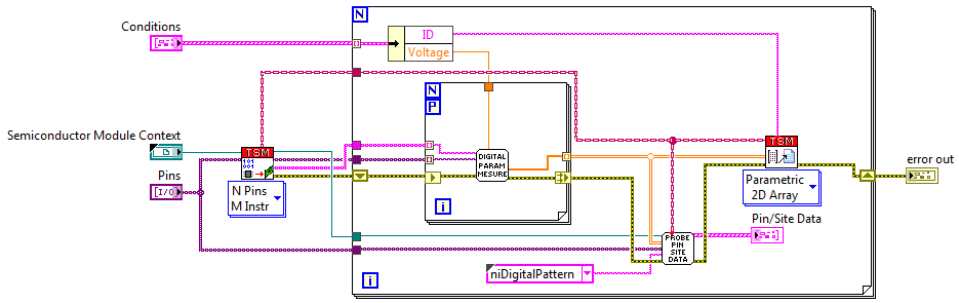
1. Query for the sessions for the power supply pin(s) to source a voltage.
2. Use a parallel For Loop to iterate on every instrument you need to call.
3. Query the sessions for the digital I/O pins on which you want to take a measurement.
4. Use a parallel For Loop to iterate on every instrument you need to call to take the measurement on all pins in all sites.
5. Use the Pin Query Context that the digital I/O pin query generates with a pin-based instance of the Publish Data VI to publish the measurement.
6. Use a parallel For Loop to iterate on every instrument you need to call to disable the power supply pins.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each digital I/O pin, as shown in the following figure:



Multiple Measurements on Multiple Pins (TSM)

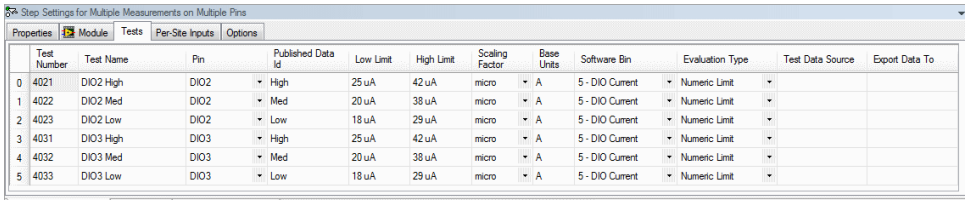
The following figure shows how to take multiple measurements on multiple pins.



This example includes the following main tasks:

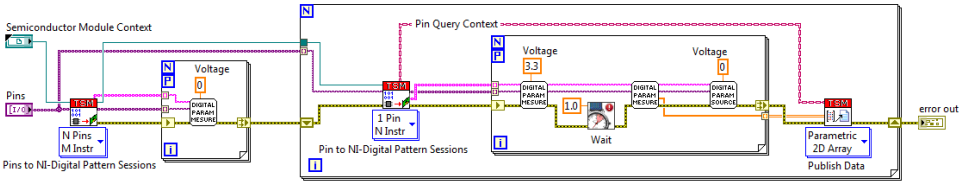
1. Query for the sessions for the pins.
2. Use a For Loop to iterate on the different measurement conditions.
3. For each condition, use a parallel For Loop to iterate on every instrument you need to call to take the measurement on all pins in all sites.
4. For each condition, use the original Pin Query Context and a unique Published Data Id with a pin-based instance of the Publish Data VI to publish the measurement for each condition. You do not need to query for sessions inside the For Loop because multiple queries will return the same sessions. You can use the same Pin Query Context to publish multiple times as long as the Published Data Id is unique.
5. Probe the measurement value for every pin and site combination. The source code for the probe is in the Probe Pin Site Data VI.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each digital I/O pin and each unique Published Data Id, as shown in the following figure:



Source All Digital Pins, Source-Wait-Measure Each Digital Pin (TSM)

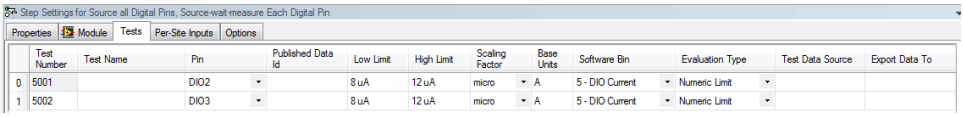
The following figure shows how to source zero volts on all digital pins and then for each digital pin, source a voltage, wait, and measure.



This example includes the following main tasks:

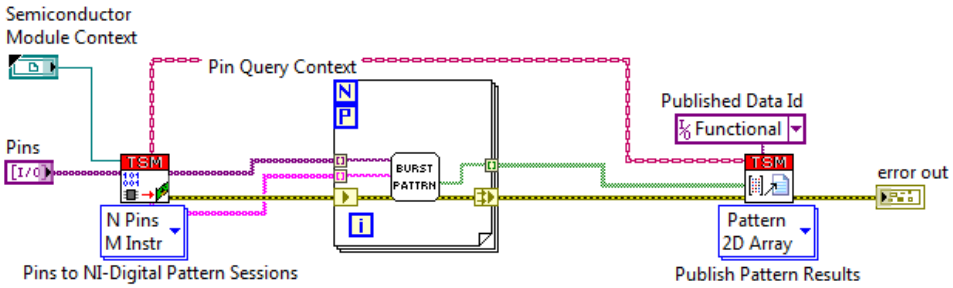
1. Query for the sessions for all test pins.
2. Use a parallel For Loop to iterate on every instrument you need to call to source zero volts on all pins.
3. Use a For Loop to iterate on every digital I/O pin.
4. For each pin, query for the session for the pin under test.
5. Use a parallel For Loop to iterate on every instrument you need to call to source a voltage, wait, and take a measurement on the pin under test.
6. Use the Pin Query Context for the pin under test with a pin-based instance of the Publish Data VI to publish the data for each pin.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each pin, as shown in the following figure:



Functional Test Using NI-Digital Pattern Driver (TSM)

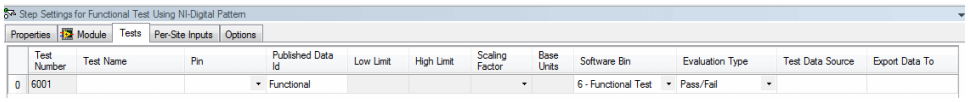
The following figure shows how to use the Digital Pattern Instrument for a multisite functional test.



This example includes the following main tasks:

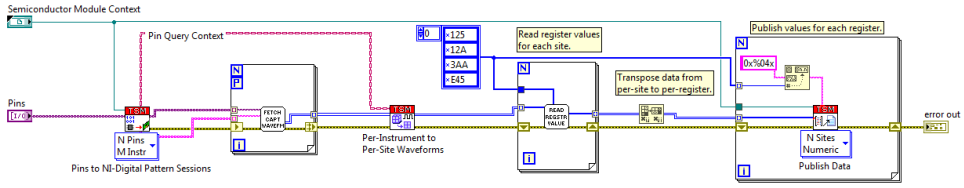
1. Query for the sessions for the digital I/O pins under test.
2. Use a parallel For Loop to iterate on every required instrument to burst a pattern.
3. Use the Publish Pattern Results VI to publish per-site Boolean results from the pattern burst.

The Semiconductor Multi Test step that calls the code module shown above contains the following test, which evaluates one Boolean result, as shown in the following figure:



Functional Test for Multiple Registers Using NI-Digital Pattern Driver (TSM)

The following figure shows how to read multiple register values from captured waveforms.

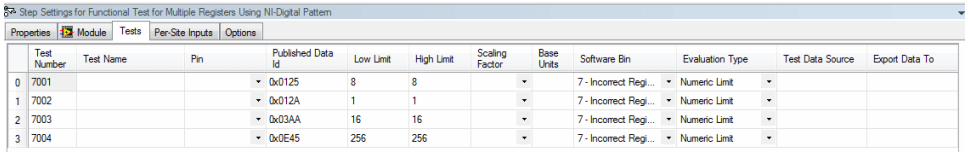


This example includes the following main tasks:

1. Query for the sessions for the digital I/O pins.
2. Use a parallel For Loop to burst patterns and fetch capture waveforms.
3. Convert the per-instrument waveform data to per-site waveforms.
4. Use a For Loop to scan each per-site waveform to obtain register values for four addresses.

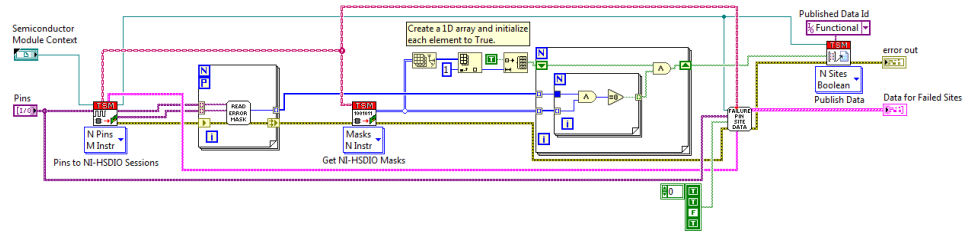
5. Transpose the 2D array so that the first dimension corresponds to sites.
6. Use a site-based instance of the Publish Data VI to publish the register value using the address as a hex string for the Published Data Id.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one numeric value per register, as shown in the following figure:



Functional Test Using NI-HSDIO Hardware Compare (TSM)

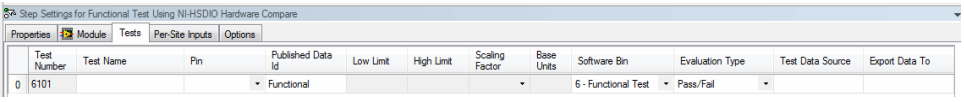
The following figure shows how to use the NI-HSDIO hardware compare engine for a multisite functional test.



This example includes the following main tasks:

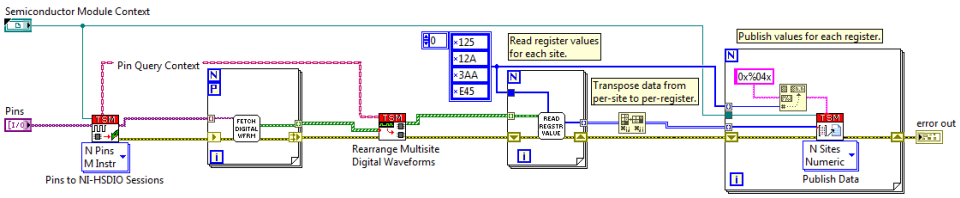
1. Query for the sessions for the digital I/O pins under test.
2. Use a parallel For Loop to iterate on every required instrument to start digital hardware compare and to obtain the cumulative error masks.
3. Use the Pin Query Context to obtain the digital per-instrument, per-site channel masks.
4. Compare the per-instrument error masks to the per-site, per-instruments masks to identify sites that failed.
5. Query for measurement data for sites that failed. The source code for the query is in the Failure Pin Site Data VI.
6. Use the Publish Data VI to publish per-site Boolean results.

The Semiconductor Multi Test step that calls the code module shown above contains the following test, which evaluates one Boolean result, as shown in the following figure:



Functional Test for Multiple Registers Using NI-HSDIO (TSM)

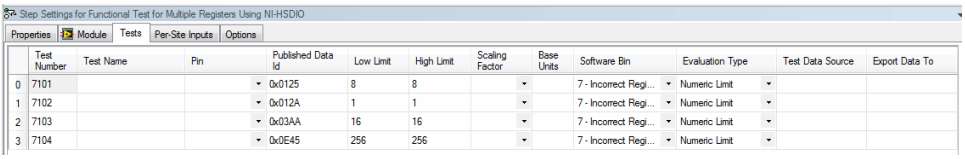
The following figure shows how to read multiple register values from a digital waveform.



This example includes the following main tasks:

1. Query for the sessions for the digital I/O pins.
2. Use a parallel For Loop to fetch the digital waveforms.
3. Rearrange the digital waveforms from per-instrument waveforms to per-site waveforms.
4. Use a For Loop to scan each per-site waveform to obtain register values for four addresses.
5. Transpose the 2D array so that the first dimension corresponds to sites.
6. Use a site-based instance of the Publish Data VI to publish the register value using the address as a hex string for the Published Data Id.

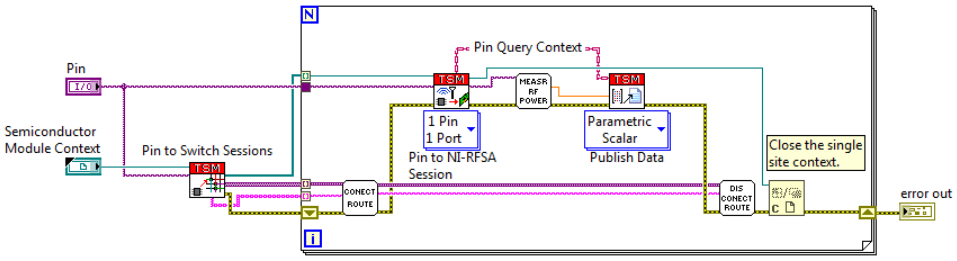
The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one numeric value per register, as shown in the following figure:



Instrument Multiplexed Across Sites (TSM)

The following figure shows how to take a measurement using an instrument connected to multiple sites through a multiplexer.

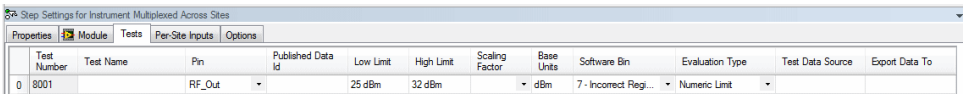
Note Run-time site order is not guaranteed to be in any particular order. Site order depends on the arrival order at the executing step.



This example includes the following main tasks:

1. Query the multiplexer sessions and routes for the multiplexed pin.
2. Use a For Loop to complete the following tasks for each multiplexed site:
    a. Connect the required multiplexer route for the site under test.
    b. Use the Semiconductor Module context created for each site to query for the instrument session for the pin.
    c. Take a measurement for a single site.
    d. Use the Pin Query Context with a pin-based instance of the Publish Data VI to publish the measurement.
    e. Disconnect the multiplexer route.
    f. Close the single site Semiconductor Module context.

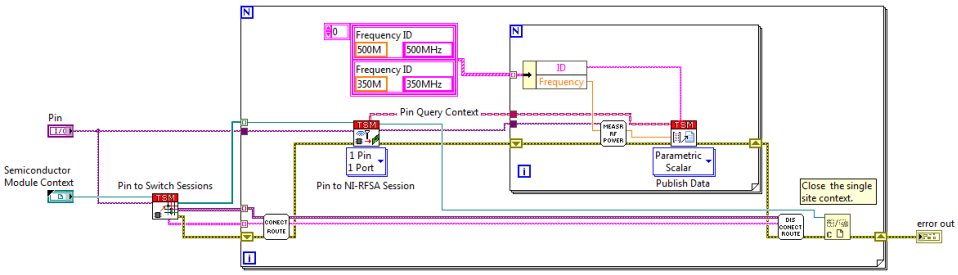
The Semiconductor Multi Test step that calls the code module shown above contains the following test, which evaluates one measurement for the multiplexed pin, as shown in the following figure:



Instrument Multiplexed Across Sites, Multiple Measurements (TSM)

The following figure shows how to take multiple measurements using an instrument multiplexed across multiple sites.

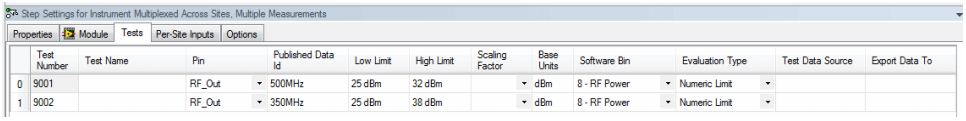
Note Run-time site order is not guaranteed to be in any particular order. Site order depends on the arrival order at the executing step.



This example includes the following main tasks:

1. Query the multiplexer sessions and routes for the multiplexed pin.
2. Use a For Loop to complete the following tasks for each multiplexed site:
    a. Connect the required multiplexer route for the site under test.
    b. Use the Semiconductor Module context created for each site to query for the instrument session for the pin.
    c. Use a For Loop to complete the following tasks for each measurement condition:
        a. Take a measurement for a single site and a single condition.
        b. Use the Pin Query Context and the unique Published Data Id with a pin-based instance of the Publish Data VI to publish the measurement.
    d. Disconnect the multiplexer route.
    e. Close the single site Semiconductor Module context.

The Semiconductor Multi Test step that calls the code module shown above contains the following tests, where each test evaluates one measurement for each measurement condition, as shown in the following figure:



Parametric I/V Instrument Panel (TSM)

Purpose

This example demonstrates how to use the custom instrument panel infrastructure to provide pin-aware debugging and control of NI-DCPower and NI-Digital SMU, power supply, and PPMU instruments during execution of a TSM test sequence.

Example File Location

`<TestStand Public>\Examples\Custom Instrument Panels\LabVIEW\Parametric I V Instrument Panel`

Installed File Location

`<TestStand Public>\Components\Modules\NI_SemiconductorModule\CustomInstrumentPanels\Parametric I V Instrument Panel`

Highlighted Features

Custom instrument panels

Major API

- NI-DCPower
- NI-Digital Pattern Driver (PPMU only)

Prerequisites

- NI-Digital Pattern Driver
- NI-DCPower

How to Use This Example

Complete the following steps to use this example.

- Open <TestStand Public>\Examples\NI_SemiconductorModule\Accelerometer\LabVIEW\Accelerometer.seq and set a breakpoint on a step in the **Main** section of the MainSequence sequence.
- Click the **Start Lot** button on the TSM toolbar to run the test program.
- When the execution stops at the breakpoint, select **Semiconductor Module»Custom Instrument Panels»Parametric I/V Instrument Panel** to launch the corresponding custom instrument panel VI.
- In the Parametric I/V Instrument Panel VI, select **Site number** or **System pins** to update the available pins list and then select a pin.
- Update the hardware settings controls to change the state of the pin. The custom instrument panel immediately commits the changes you make to the pin you selected. The hardware settings controls display the current state of the pin.
  - The custom instrument panel VI uses the configured settings to continuously run a software-timed current (I) and voltage (V) measurement on the selected pin, performing the measurement approximately once every 500ms.
  - The **Current** and **Voltage** charts display the measurement results and provide a historical view of the state of the selected pin.

Note You can click the **Pause** button to pause the continuous measurements. Clicking the **Pause** button does not change the state of the selected pin and pauses only the measurements the custom instrument panel VI performs on the selected pin.

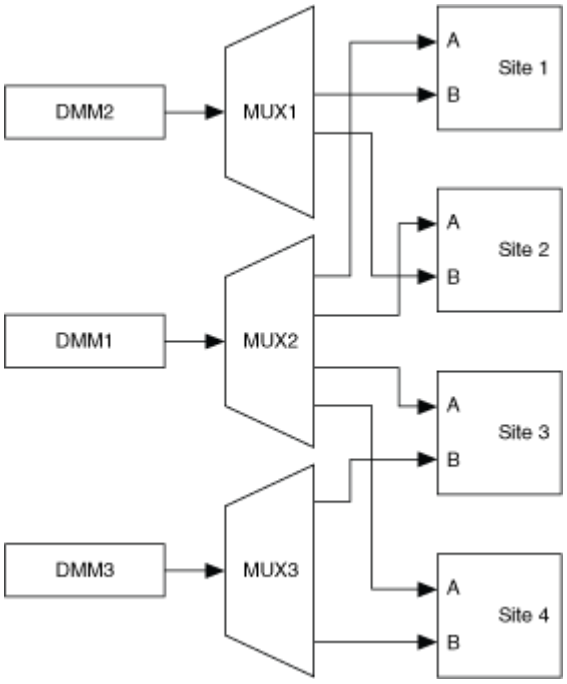
- Click the **Close (X)** button in the title bar of the Parametric I/V Instrument Panel VI to end the execution of the custom instrument panel VI.

Switching (TSM)

Purpose

This example demonstrates a test program that uses four sites and three shared instruments, as shown in the following figure. The test program shares the `DMM1` instrument across all four sites and shares the `DMM2` and `DMM3` instruments across two sites each. All three instruments contain only one channel routed to each site using a switch (MUX).

Note Run-time site order is not guaranteed to be in any particular order. Site order depends on the arrival order at the executing step.



Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Switching\LabVIEW\Switching.seq`

Highlighted Features

- Multisite
- Pin map

- Switching

Major API

TSM Code Module API

Switch Executive API

Prerequisites

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- You must have NI-Switch 15.0 or later installed.
- You must have Switch Executive 2015 or later installed.
- This example uses the Batch process model.

You must also configure the switch instruments and route in Measurement & Automation Explorer (MAX) to be available through Switch Executive Virtual Devices. This example includes the `SwitchingConfiguration.nce` NI Configuration Export File to import an example configuration into MAX. To import the file, open MAX, select **File»Import**, and browse to the `SwitchingConfiguration.nce` file in the resulting dialog box.

How to Use This Example

Complete the following steps to use this example.

1. Complete the following steps to review the pin map file associated with the test program.
    a. Select **Semiconductor Module»Edit Pin Map File**, click the **Edit Pin Map File** button on the TSM toolbar, or launch the Test Program Editor and click the **Open file for edit** button located to the right of the Pin Map File Path display on the Pin Map panel to open the `<TestStand Public>\Examples\NI_SemiconductorModule\Switching\LabVIEW\Switching.pinmap` file in the Pin Map Editor.

       The pin map file defines the following information:

- Two DUT pins named A and B, respectively.
- Three multiplexer switch instruments named Mux1, Mux2, and Mux3, each with a multiplexerTypeId attribute value of NISimulatedMultiplexer.
- Three DMM instruments named DMM1, DMM2, and DMM3, each with one channel.
- Four sites on the tester.
- A set of multiplexed connections, in which each connection specifies an instrument and instrument channel and contains a list of connection pin routes. Each route specifies a site, route name, multiplexer, and pin to which it is connected.

2. Select each step in the MainSequence sequence and click the Tests tab on the Step Settings pane to review the tests each step performs.

   The Semiconductor Multi Test step type uses the information in the pin map to determine which sites, instruments, and instrument channels to use for a particular test. The first step uses DMM1, which is shared across all four sites, to test pin A on each site. The second step uses DMM2 and DMM3, which are each shared across two sites, to test pin B on each site.
3. Click the Options tab on the Step Settings pane to review the DUT pins specified for each step and the multisite execution that the step uses.

   The first step specifies only DUT pin A. The step uses the pin map to determine that only one instrument is connected to pin A for each site and that a switch routes the channel to each site. With this information, the step determines that all four sites must run in a single thread and only one instance of the code module executes, as shown in the Multisite Execution Diagram.

   The second step specifies only DUT pin B. The step uses the pin map to determine that two instruments are available and each connects to DUT pin B on independent sites. With this information, the step determines that it is able to run the code module in two independent threads, as shown in the Multisite Execution Diagram.

4. On the Sequences pane, select the `ProcessSetup` sequence. The steps in this sequence initialize the custom instruments and switches.
5. On the Sequences pane, select the `ProcessCleanup` sequence. The steps in this sequence close instrument and switch sessions.
6. Open `<TestStand Public>\Examples\NI_SemiconductorModule\Switching\LabVIEW\Switching.lvproj` to review the code modules the test program uses. The Example Test Code VI demonstrates a test code module that uses multiplexed routes. In the VI, each iteration of the For Loop tests one site. The tests for each site execute serially in a For Loop because the tests share an instrument. Do not enable loop iteration parallelism in the For Loop.

   Note The For Loop must also close each reference in the array of Semiconductor Module contexts because the references were generated in LabVIEW instead of in TestStand and will not be closed automatically.

7. Click the **Start Lot** button on the TSM toolbar to run the test program.

Initializing Switch Sessions (TSM)

In the `ProcessSetup` callback sequence of a test program, you must initialize all instrument sessions and switch sessions and store them in the Semiconductor Module context. Refer to the Add Switch Sessions VI in the Switching example for an example of how to initialize a set of switch sessions.

Pass `NISimulatedMultiplexer` for the **Multiplexer Type ID** parameter of the Get All Switch Names VI to obtain an array of switch instrument names from the pin map. In a For Loop, iterate through each name in the Switches Names array and create a session for each switch instrument. In the same For Loop, use the Set Switch Session VI to associate a switch name with a switch session in the Semiconductor Module context.

Closing Switch Sessions (TSM)

In the `ProcessSetup` callback sequence of a test program, you must close all instrument sessions and switch sessions stored in the Semiconductor Module

context. Refer to the Close Switch Sessions VI in the Switching example for an example of how to clean up switch sessions.

Pass `NISimulatedMultiplexer` for the **Multiplexer Type ID** parameter of the Get All Switch Sessions VI to obtain an array of switch sessions from the pin map. In a For Loop, iterate through each session in the Session Data array and close the session. Because the session data is an array of variant data types to allow for any session type, use the Variant To Data function inside the For Loop to convert the variant data type to the correct switch session type.

Configuring Switches with Switch Executive (TSM)

NI recommends using Switch Executive to configure complex switching routes. Switch Executive provides a user interface to configure routes and route groups in Measurement & Automation Explorer. Refer to the **Switch Executive Help** for more information about configuring switch routes.

You can deploy the route configuration to production test systems by exporting the virtual device as part of an NI Configuration Export File that Measurement & Automation Explorer generates and then including the file in a TestStand deployment.

Code Modules with Multiplexed Routes (TSM)

In a code module you call from a Semiconductor Multi Test step, you must connect DUT pins connected to multiplexers to the corresponding switch route before you can perform a test on that DUT pin. At the end of the test, disconnect the route. Refer to the Switching Example Test Code VI in the Switching example for an example of how to perform a measurement on multiplexed connections.

Note Do not use the TestStand IVI Switching mode or Switch step properties with TSM test programs because the mode and properties are not multisite aware.

In the code module, pass `NISimulatedMultiplexer` for the **Multiplexer Type ID** parameter of the Pin to Switch Sessions VI to obtain an array of switch sessions, switch route names, and Semiconductor Module contexts for each site in the subsystem. Each element in the Semiconductor Module contexts array represents a

single site on which the test code must execute serially. Use the Semiconductor Module context object to query the pin map and publish data for the site. In a For Loop, use the switch session and the corresponding route name to connect the switch route. Because the session data is an array of variant data types to allow for any session type, use the Variant To Data function inside the For Loop to convert the variant data type to the correct switch session type. Because auto-indexing is enabled on the For Loop, it ensures that the switch session, switch route name, and Semiconductor Module context correspond to each other for each iteration of the For Loop.

Note Do not enable parallel iterations on a For Loop to iterate through the elements of the Semiconductor Module contexts array because the sites share common instruments, and the iterations must execute serially.

After you connect the route, call the Pin Query VI inside the For Loop to query the pin map for the instrument session. Each iteration of the For Loop uses a separate instance of the Semiconductor Module context from the array the Pin To Switch Sessions VI returns. Use the instrument session to perform the measurement on the pin. Use the Publish Data VI to publish the measurement by passing the Pin Query Context the Pin Query VI returns to the Publish Data VI. After you publish the measurement, you must close the Semiconductor Module context reference because the reference was generated in LabVIEW, and the code module must manage the reference. You do not need to close the reference for the Semiconductor Module context that you pass as a parameter from TestStand.

Before exiting the For Loop, use the same switch session and switch route name to disconnect the switch route.

Part Average Testing Examples (TSM)

The `<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing` directory contains the following examples:

- Part Average Testing with LabVIEW
- Part Average Testing with .NET

Example Part Average Testing Plug-In

The <TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In directory contains a part average testing (PAT) plug-in that you can use as a starting point for custom PAT plug-ins you create.

Note By default, TSM uses data that cause all tests to pass when you run in Offline Mode. The PAT example is configured to override this default behavior and use the published values for tests. When creating a custom PAT plug-in from the PAT example, modify the Numeric Limit Test or Pass/Fail Test property settings to specify which test values to use in Offline Mode.

The example PAT plug-in consists of the following files:

- PartAverageTestingCallbacks.seq—PAT callback sequence file
- PartAverageTestingAlgorithmExample.dll—.NET assembly that contains code modules the callback sequence file calls

Opening the Part Average Testing Example.seq file in the TestStand Sequence Editor automatically copies the example PAT plug-in files from the <TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In directory to the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory. Closing the Part Average Testing Example.seq file removes the example PAT plug-in files from the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory.

See Also

Part Average Testing

Part Average Testing with LabVIEW (TSM)

Purpose

This example demonstrates how to enable and perform part average testing (PAT) in a test program.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\LabVIEW\Part Average Testing Example.seq`

Highlighted Features

- TSM PAT plug-in architecture
- Test Program Editor PAT Algorithm Settings panel
- Semiconductor Multi Test Part Average Testing tab
- Perform Part Average Testing step
- Static PAT limits file
- OnSiteTestingComplete callback sequence

Major API

TSM Application API

Prerequisites

Note To perform part average testing, you must place the PAT plug-in files in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule` directory. Opening the `Part Average Testing Example.seq` file in the TestStand Sequence Editor automatically copies the example PAT plug-in files from the `<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In` directory to the `<TestStand Public>\Components\Callba`

cks\NI_SemiconductorModule directory. Closing the Part Average Testing Example.seq file removes the example PAT plug-in files from the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory.

- You must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- You must have NI-DCPower 15.1 or later installed. You do not need an NI-DCPower instrument because this example uses simulated instrument sessions.
- This example uses the Batch process model.

Note You can view the test program in the TestStand Sequence Editor and partial code modules in LabVIEW without the required NI instrument drivers installed. Install the drivers to view the full code modules in LabVIEW. Visit ni.com/info and enter the Info Code rddrau to access the latest software drivers and updates.

How to Use This Example

Complete the steps in the following sections to learn about TSM PAT plug-ins, to access and modify the PAT settings, and to enable, disable, and configure part average testing for individual tests.

1. Select **Semiconductor Module»Edit Test Program: Part Average Testing Example.seq** or click the **Edit Test Program: Part Average Testing Example.seq** button on the TSM toolbar to launch the Test Program Editor for the sequence file.
2. Select the PAT Algorithm Settings panel and review the following settings the test program specifies for the PAT plug-in to use.
   - **Disable Static Part Average Testing**—Disables all static part average testing for the test program.

- **Disable Dynamic Part Average Testing**—Disables all dynamic part average testing for the test program.
- **Static PAT Limits File**—Specifies a relative path to a file that contains static limits for all tests enabled for static part average testing in the test program. The example PAT algorithm uses the `IPartAverageTestingStaticLimitLoader.LoadStaticLimits` method in the TSM Application API to load a static limits file that uses the same structure as a TSM test limits file. To read and use limits from a different file format, you must implement a custom file reader and a custom data structure to store the limits.
- **Statistics Type**—Specifies the type of statistics to use to calculate dynamic PAT limits.
- **Window Size**—Specifies how many measurements of previously tested DUTs to use to calculate the dynamic limits for the DUT currently being tested.
- **Early Part Count**—Specifies how many parts in a lot must complete testing before considering the computed dynamic limits statistically valid. DUTs that complete testing before this value are called early parts. This example PAT algorithm sets the dynamic limits for early parts to the test program limits.

3. Select the Continuity step in the `MainSequence` sequence. The example will perform part average testing for the tests in this step when it executes the Perform Part Average Testing step.
4. Click the **Part Average Testing** tab.
5. Add a checkmark to or remove a checkmark from the checkboxes in the **Enable Dynamic PAT** and **Enable Static PAT** columns.
6. Set test numbers, test names, and software fail bins for the tests with dynamic or static part average testing enabled. Bins must be defined in the bin definitions file.
7. Enable and display the Debug Test Results Log by placing a checkmark in the **Enabled** column and in the **Display** column of the Result Processing dialog box.

8. Ensure that you meet the prerequisites, then click the **Single Test** button on the TSM toolbar to test a single part on each site.
9. Click the **End Lot** button.
10. Click the **Report** pane of the Execution window and review the Debug Test Results Log to see the PAT tests that executed during the `MainSequence` sequence.

See Also

Part Average Testing

Part Average Testing with .NET (TSM)

Purpose

This example demonstrates how to enable and perform part average testing (PAT) in a test program.

Example File Location

`<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\DotNET\Part Average Testing Example.seq`

Highlighted Features

- TSM PAT plug-in architecture
- Test Program Editor PAT Algorithm Settings panel
- Semiconductor Multi Test Part Average Testing tab
- Perform Part Average Testing step
- Static PAT limits file
- OnSiteTestingComplete callback sequence

Major API

TSM Application API

Prerequisites

Note To perform part average testing, you must place the PAT plug-in files in the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory. Opening the Part Average Testing Example.seq file in the TestStand Sequence Editor automatically copies the example PAT plug-in files from the <TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In directory to the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory. Closing the Part Average Testing Example.seq file removes the example PAT plug-in files from the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory.

- You must have NI-DCPower 15.1 or later installed. You do not need an NI-DCPower instrument because this example uses simulated instrument sessions.
- You must have .NET 4.0 support for the NI-DCPower .NET Class Libraries 1.1 or later installed.
- This example uses the Batch process model.

Note You can view the test program in the TestStand Sequence Editor and code modules in a C# source code editor without the required NI instrument drivers installed. Visit ni.com/info and enter the Info Code NETAPIdriversupport for information about the available NI .NET APIs and the versions of the NI drivers each supports.

How to Use This Example

Complete the steps in the following sections to learn about TSM PAT plug-ins, to access and modify the PAT settings, and to enable, disable, and configure part average testing for individual tests.

1. Select **Semiconductor Module»Edit Test Program: Part Average Testing Example.seq** or click the **Edit Test Program: Part Average Testing Example.seq** button on the TSM toolbar to launch the Test Program Editor for the sequence file.
2. Select the PAT Algorithm Settings panel and review the following settings the test program specifies for the PAT plug-ins to use.
   - **Disable Static Part Average Testing**—Disables all static part average testing for the test program.
   - **Disable Dynamic Part Average Testing**—Disables all dynamic part average testing for the test program.
   - **Static PAT Limits File**—Specifies a relative path to a file that contains static limits for all tests enabled for static part average testing in the test program. This example PAT algorithm uses the `IPartAverageTestingStaticLimitLoader.LoadStaticLimits` method in the TSM Application API to load a static limits file that uses the same structure as a TSM test limits file. To read and use limits from a different file format, you must implement a custom file reader and a custom data structure to store the limits.
   - **Statistics Type**—Specifies the type of statistics to use to calculate dynamic PAT limits.
   - **Window Size**—Specifies how many measurements of previously tested DUTs to use to calculate the dynamic limits for the DUT currently being tested.
   - **Early Part Count**—Specifies how many parts in a lot must complete testing before considering the computed dynamic limits statistically valid. DUTs that complete testing before this value are called early parts. This example PAT algorithm sets the dynamic limits for early parts to the test program limits.

3. Select the Continuity step in the `MainSequence` sequence. The example will perform part average testing for the tests in this step when it executes the Perform Part Average Testing step.
4. Click the **Part Average Testing** tab.
5. Add a checkmark to or remove a checkmark from the checkboxes in the **Enable Dynamic PAT** and **Enable Static PAT** columns.
6. Set test numbers, test names, and software fail bins for the tests with dynamic or static part average testing enabled. Bins must be defined in the bin definitions file.
7. Enable and display the Debug Test Results Log by placing a checkmark in the **Enabled** column and in the **Display** column of the Result Processing dialog box.
8. Ensure that you meet the prerequisites, then click the **Single Test** button on the TSM toolbar to test a single part on each site.
9. Click the **End Lot** button.
10. Click the **Report** pane of the Execution window and review the Debug Test Results Log to see the PAT tests that executed during the `MainSequence` sequence.

See Also

Part Average Testing

Test Steps and Flow (TSM)

A test step is an instance of the Semiconductor Multi Test step type or a custom step type based on the Semiconductor Multi Test step type that performs one or more parametric or functional tests. A test step calls a code module implemented in LabVIEW or .NET to control the instrumentation on the tester, take measurements from the DUT, and pass measurement values back to the Semiconductor Multi Test step. The step performs parametric and functional tests using the measurement values obtained from the code module. The Main sequence of a test program sequence file defines the test steps to execute and the order in which to execute

them. You can use standard TestStand control flow features, such as preconditions and Flow Control step types, to control the test flow.

To skip subsequent tests when a test fails, enable the **Stop Performing Tests after First Failure** option on the Options tab of the Semiconductor Multi Test step in the sequence editor and also set the **On Step Failure** sequence option to **Goto Cleanup** on the General tab in the Sequence Properties dialog box. To skip one or more tests based on specific conditions, use the Get Test Information step to save the condition in a local variable and use an expression that refers to the variable in a precondition expression or in the condition expression of an If step. Skipping a test step execution skips all tests associated with the test step.

Note You cannot use multiple Semiconductor Multi Test steps or Semiconductor Action steps configured to use multiple threads in While loops, in Do While loops or in For loops that use the Custom Loop option when performing multisite testing. The steps report a run-time error in these situations. Use other types of loops instead, such as For loops that use the Fixed Number of Iterations option.

Programming Multiple Test Flows in a Test Program

You can execute multiple test flows from a single test program. A test program can contain multiple configurations to specify values for test conditions that can control execution flow. Use the Test Program Editor to specify configurations and test condition values for a test program.

TSM sets available test configurations when you configure a lot for execution. You can use the values of test conditions to control the flow of the sequence when used as inputs to TestStand Flow Control step types. Use the Get Test Information step type to obtain the value of a test condition at run time. You can specify the TestFlowId standard lot setting as a standard test condition that identifies the test flow in an STDF log file.

Performing Tasks after all Tests Complete

TSM assigns a bin to the DUT after the `MainSequence` sequence executes. Therefore, if you need to perform tasks after TSM has assigned a bin to the DUT, you must perform these tasks in the OnSiteTestingComplete callback sequence. If not, you can perform them in the Cleanup step group of the `MainSequence` sequence. If you do use the Cleanup step group for these tasks and your test program uses part average testing, you should precede the tasks with a Perform Part Average Testing step, so that you can ensure that they run after all part average testing is completed. If you don't use a Perform Part Average Testing step, TSM performs part average testing after executing all step groups of the `MainSequence` sequence.

In general, do not use the PostUUT Model callback because TestStand calls this callback after TSM sends the end-of-test (EOT) notification to the handler or prober. As a result, you cannot perform operations on a DUT in the `PostUUT` callback.

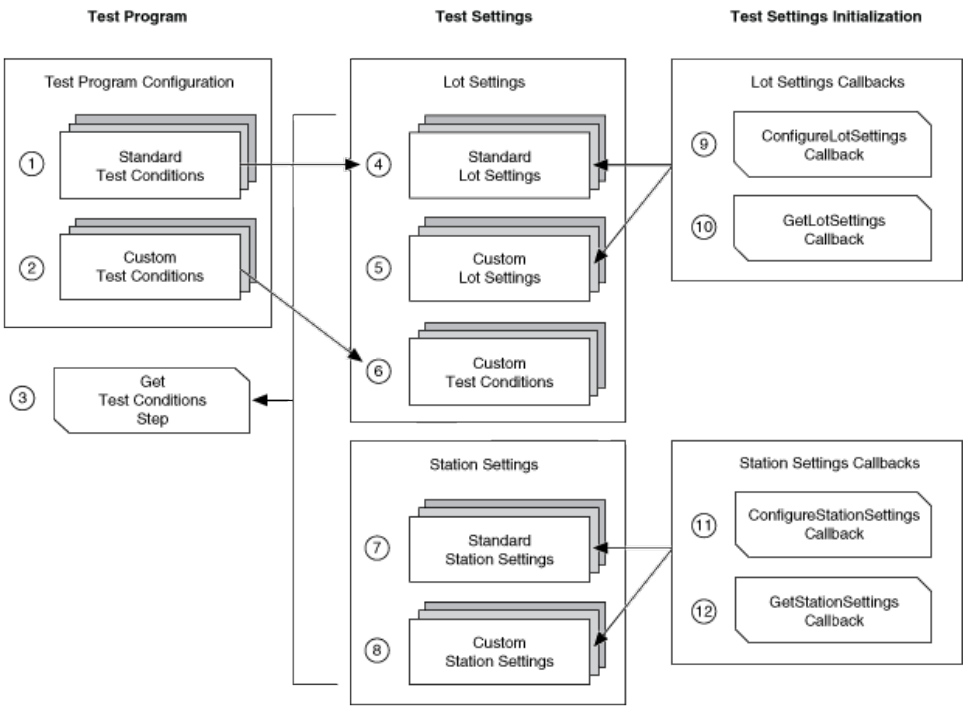
Test Settings Relationships (TSM)

TSM uses collections of settings to provide information about a test lot or test station. Lot settings and test conditions provide information about a test lot, and station settings provide information about the test station.

A semiconductor test program can provide values for some settings and can use configurations to create new settings specific to the test program. The test program can then use the Get Test Information step to access those values in a sequence.

You can use the Configure Lot Settings dialog box and the Configure Station Settings dialog box to assign values for these settings. You can also use the GetLotSettings and GetStationSettings callback sequences to programmatically assign the values. You can customize both methods by modifying the ConfigureLotSettings, GetLotSettings, ConfigureStationSettings, or GetStationSettings callback sequences to provide routines for obtaining these values.

The following figure shows the relationship among the test program, test settings, and the methods of initialization.



1. **Standard Test Conditions**—Specifies lot information that is standard for all test programs for the current test program configuration. These properties map to a subset of standard lot settings that correspond to some fields of the Master Information Record (MIR) of the Standard Test Data Format (STDF) version 4 specification. The standard test conditions also include other properties specific to TSM. Standard test conditions defined in the test program overwrite the value of the corresponding standard lot setting at run time. Use the Configuration Definition panel in the Test Program Editor to add standard and custom test conditions to a test program.
2. **Custom Test Conditions**—Specifies lot information that is specific to the test program for the current test program configuration. Test engineers define these properties, which are loaded into the lot settings at run time. Use the Configuration Definition panel in the Test Program Editor to add standard and custom test conditions to a test program.
3. **Get Test Information Step**—Use the Get Test Information step to obtain the values for lot settings, station settings, STS state, execution data, and custom test conditions. Store the value of the items in TestStand local variables so that any step in the sequence can access the values.

4. **Standard Lot Settings**—Specifies lot information for the current test lot. These properties correspond to some fields of the Master Information Record (MIR) of the Standard Test Data Format (STDF) version 4 specification. The standard lot settings also include other properties specific to TSM. TSM exposes the lot settings to the test program by attaching a copy of the lot settings to the execution run-time variables (`RunState.Execution.RunTimeVariables.NI.SemiconductorModule.LotSettings`). Use the ConfigureLotSettings callback sequence to prompt an operator to manually enter lot information in a dialog box or to use another mechanism that requires user input. A subset of these properties are also available to test engineers as standard test conditions.
5. **Custom Lot Settings**—Specifies lot information for the current test lot. Test architects define these properties, which are available to all test programs on the station. TSM exposes the lot settings to the test program by attaching a copy of the lot settings to the execution run-time variables (`RunState.Execution.RunTimeVariables.NI.SemiconductorModule.LotSettings`).
6. **Custom Test Conditions**—TSM copies custom test conditions, as described in Item 2, into lot settings when execution begins.
7. **Standard Station Settings**—Specifies configuration options for the tester that apply to all test lots and that persist during restart and shutdown operations, such as handler or prober configuration or data logging preferences. These properties correspond to some fields of the Master Information Record (MIR) of the Standard Test Data Format (STDF) version 4 specification. The standard station settings also include other properties specific to TSM. Use the ConfigureStationSettings callback sequence to prompt a test engineer or technician to manually configure station settings in a dialog box or to use another mechanism that requires user input.
8. **Custom Station Settings**—Specifies custom configuration options for the tester that apply to all test lots and that persist during restart and shutdown operations. The test architect defines these properties, which are available to all test programs on the station. Use the ConfigureStationSettings callback sequence to prompt a test engineer or technician to manually configure

station settings in a dialog box or to use another mechanism that requires user input.

9. **ConfigureLotSettings Callback**—The **Semiconductor Module»Configure Lot** menu item, the **Configure Lot** button on the TSM toolbar in the TestStand Sequence Editor, and the **Configure Lot** button in the default TSM operator interface call the ConfigureLotSettings callback sequence to obtain the settings for the current test lot. Use the `ConfigureLotSettings` callback sequence to prompt an operator to manually enter lot information in a dialog box or to use another mechanism that requires user input.
10. **GetLotSettings Callback**—TSM calls the GetLotSettings callback sequence to programmatically obtain lot settings without requiring much, if any, operator interaction when execution begins. You can override this callback sequence to customize the behavior when TSM attempts to determine lot settings values at run time.
11. **ConfigureStationSettings Callback**—The **Semiconductor Module»Configure Station** menu item, the **Configure Station** button on the TSM toolbar in the TestStand Sequence Editor, and the **Configure Station** button in the default TSM operator interface call the ConfigureStationSettings callback sequence to obtain the settings for the current test station. Use the `ConfigureStationSettings` callback sequence to prompt a test engineer or technician to manually configure station settings in a dialog box or to use another mechanism that requires user input.
12. **GetStationSettings Callback**—TSM calls the GetStationSettings callback sequence to programmatically obtain station settings without requiring much, if any, test engineer or technician interaction when execution begins. You can override this callback sequence to customize the behavior when TSM attempts to determine station settings values at run time.

Execution Timing Overview (TSM)

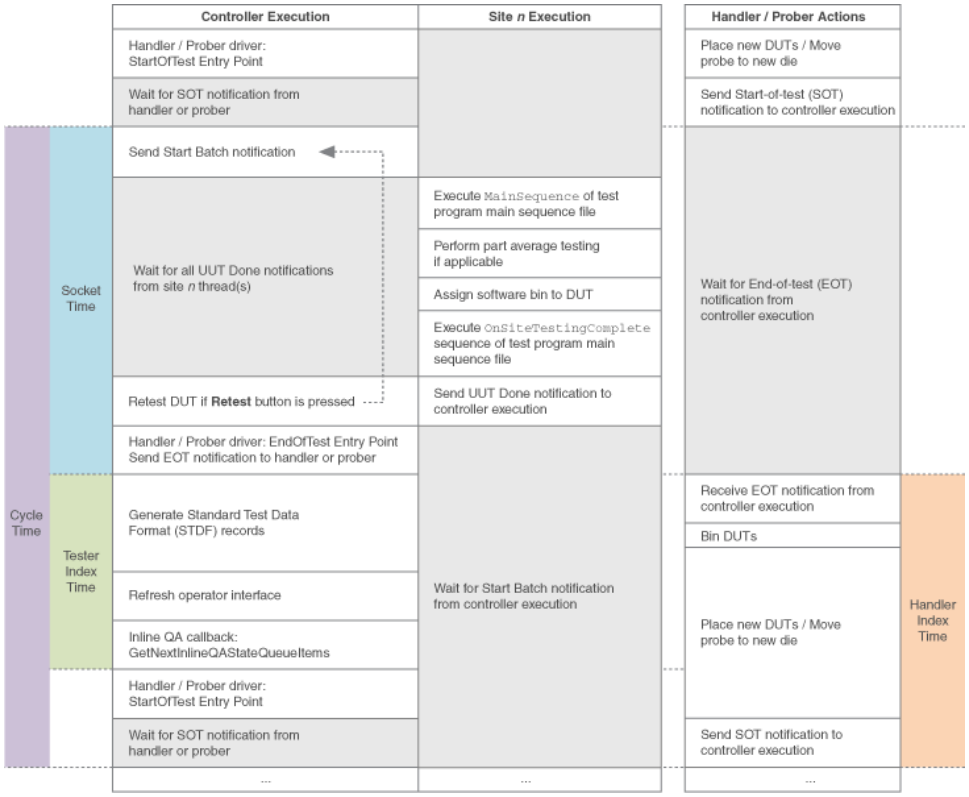
The following diagram illustrates how TSM coordinates test program execution and handler/prober actions.

Note Executing a test program in a TSM operator interface is typically faster than executing the test program in the TestStand Sequence Editor development environment.

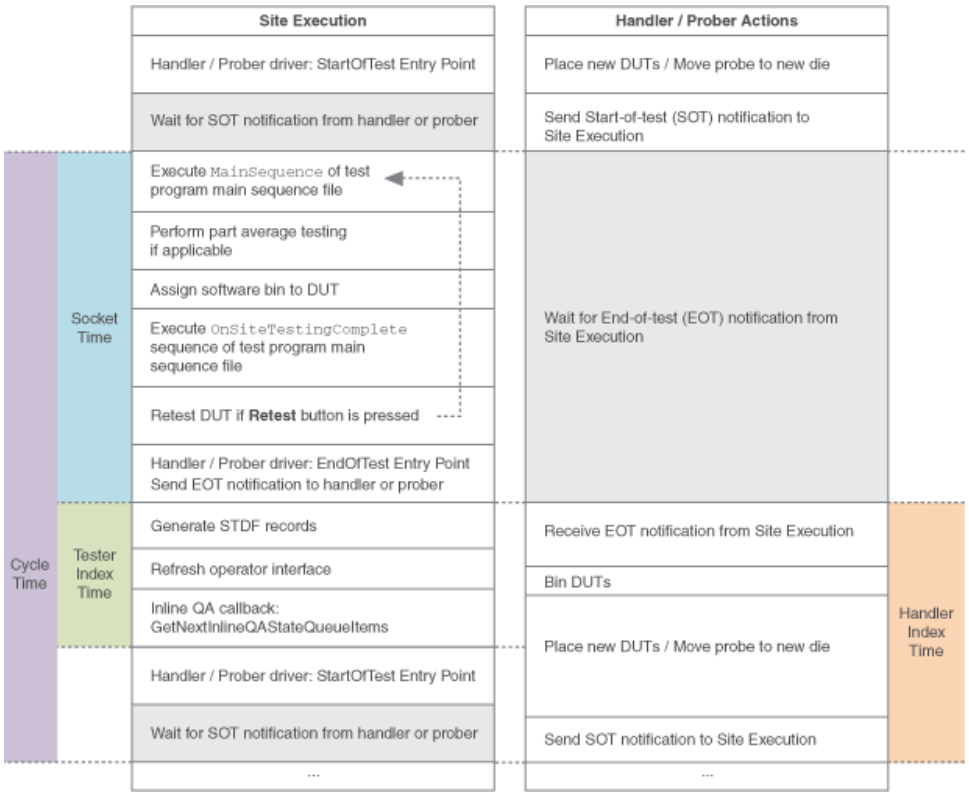
Legend
- Wait actions
- Cycle time
- Socket time
- Tester index time
- Handler/Prober index time

Batch Model

When you execute tests using the Batch process model, a controller thread controls the core tester software functionality and sends notifications to and receives notifications from a separate thread for each site execution.



Sequential Process Model



See Also

Handler/Prober Driver Entry Points

Performing Inline Quality Assurance Testing

Measuring Performance (TSM)

Recommended TSM Test Program Structure and Filenames

Use the following directory and file naming recommendations to organize test programs and associated files. A consistent structure can make the test program easier to maintain and deploy.

Directory

Create a separate directory that contains the following files and subdirectories for each test program:

- Test program sequence file
- **Code Modules**—Contains tests and other necessary program-specific code.
- **Supporting Materials**—Contains the following files and subdirectories:
  - Digital pattern project file (if applicable)
  - **Bin Definitions**—Contains bin definitions file.
  - **Pin Maps**—Contains pin map file(s).
  - **Calibration Data**—Contains calibration configuration and/or data files (if applicable). Place any program-specific code in the Code Modules directory.
  - **Specifications**—Contains specification files.
  - **Digital**—Contains data files for digital tests, including the following files:
    - Pattern files
    - Timing files
    - Levels files
    - Source and capture waveform files
  - **Offline Mode Configurations**—Contains Offline Mode system configuration files.
- **Limits**—Contains the limits files for each test program configuration.

Note Create only the files and directories you need for the test program.

Filenames

For each test program, use a unique filename that identifies the part the test program tests. For the corresponding pin map, bin definitions, and digital pattern project files, use the same base filename as the test program sequence file. For

example, use `Part123.seq` for a test program sequence file and `Part123.pinmap`, `Part123.bins` and `Part123.digiproj` for the associated pin map, bin definitions, and digital pattern project files, respectively.

Mapping DUT Pins to Instrument Channels (TSM)

A semiconductor test program must communicate information from the tester instrumentation to the DUT to which the instrumentation is connected. To handle this communication in a test program, test engineers must consider the following requirements for instrument sessions and channels for all resources in the test system:

- Develop test program code that uses the names of actual connections on the DUT (DUT pins) to refer to channels on the instrument.
- Manage a large number of instrument channels.
- Scale test code to test multiple sites in parallel to improve tester efficiency.
- Support multiple types of instruments because different instruments might use channels and sessions in different ways.

Typically, instrument driver software provides test engineers with software tools for communicating with the DUT in terms of the instrument channels and sessions. However, instrument drivers have no information about the actual pins on the DUT. To develop test code that uses DUT pin names, a test engineer must use a pin map to associate each DUT pin name with the name, channel, and session of the instrument connected to that particular DUT pin.

TSM Implementation

Use the Pin Map Editor to view, create, modify, and save pin map files. The pin map XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\PinMap.xsd`, defines the structure for a pin map file. Use the **Pin Map File Path** control on the Pin Map panel of the Test Program Editor to specify the pin map file to use with the test program.

The pin map file can support multiple test sites and multiple instrument types.

In addition, use the TSM Code Module API to query instrument names, obtain all open sessions, and translate DUT pin names or pin groups to instrument channels and sessions. The Semiconductor Multi Test and the Semiconductor Action steps create the `SemiconductorModuleContext` object that you can pass to a LabVIEW or .NET code module to use the TSM Code Module API. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system.

Natively Supported and Custom Instruments (TSM)

The TSM natively supports the following types of instruments for LabVIEW and .NET:

NI Instrument	Pin Map Instrument Type	Instrument Driver and Pin Map API Support	Preferred Method of Interacting with Instrument
Digital Pattern Instrument	NIDigitalPatternInstrument	NI-Digital Pattern	Digital Pattern Editor
DC Power Supply or Source-Measure Unit (SMU)	NIDCPowerInstrument	NI-DCPower	Digital Pattern Editor
Data Acquisition and Signal Conditioning (DAQ)	NIDAQmxTask	NI-DAQmx	—
Digital Multimeter (DMM)	NIDmmInstrument	NI-DMM	InstrumentStudio
Arbitrary Waveform and Function Generator	NIFGenInstrument	NI-FGEN	InstrumentStudio
High-Speed Digitizer	NIScopeInstrument	NI-SCOPE	InstrumentStudio
High-Speed Digital I/O	NIHSDIOInstrument	NI-HSDIO	—
RF Signal Analyzer	NIRFSAInstrument	NI-RFSA NI-RFmx	RFmx SFP
RF Signal Generator	NIRFSGInstrument	NI-RFSG	RFmx SFP
Vector Signal Transceiver	NIVSTInstrument	NI-RFSA NI-RFSG NI-RFmx FPGA (LabVIEW only)	RFmx SFP

RF Port Module	NIRFPMInstrument	NI-RFPM	RFmx SFP
Relay Driver Module	NIRelayDriverModule	NI-SWITCH	Digital Pattern Editor

Notes

- The NI TestStand 2016 Semiconductor Module and later natively support digital pattern instruments that use the NI-Digital Pattern Driver and legacy digital waveform instruments that use the NI-HSDIO driver, such as the PXIe-6556. Use the TSM Code Module API that corresponds to the type of digital instrument the test system includes.
- Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation**, for example, `HSD_657x_C2_S03`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), and `SlotLocation` uses double digits to identify the slot location (`Sxx`).

Custom Instruments

You can use other types of instruments in a pin map by using the generic `<Instrument>` tag in the pin map file. Refer to the example sequence files, pin map files, and LabVIEW code module VIs located in the `<TestStand Public>\Examples\Custom Instruments` directory for examples of using TSM pin map files and VIs to perform tests using instruments that TSM does not natively support.

You can create custom site and pin aware instrument panel VIs to control and measure pins during test program execution at a breakpoint, which can be useful during test program development and troubleshooting.

Model-Based Instruments (TSM)

Use a Model-Based Instrument to specify pin map channels and instrument properties so you can connect TSM to a third-party instrument. Unlike Custom Instruments, Model-Based Instruments can be reused across test programs and pin maps.

A Model-Based Instrument is defined in an instrument model description file. The **instrument model description file** is an XML file that follows the Automatic Test Markup Language (ATML) schema, the standard for interfacing test system components using XML. The instrument model description file includes general information, details for connection components (channels, ports, resources), and configuration properties of the instrument and its resources. Reuse an instrument description file across multiple test programs and pin maps.

Instrument model description files installed by NI are saved in the Instrument Model library, located in individual subdirectories under the `<Program Files>\National Instruments\Shared\InstrumentLibrary\SystemDescriptions\ModelDescriptions` directory.

You can also define your own Model-Based Instrument. Create the following directory and save your model description file in it: `<Public>\Documents\National Instruments\InstrumentLibrary\SystemDescriptions\ModelDescriptions`.

The specific connection components of a Model-Based Instrument are defined in the model description file for all instances of the model. However, in individual systems, connection components can have different names, or aliases. These aliases correspond one-to-one with the names in the model description file. This mapping is defined in a system description file. When the mapping is defined in a system description file, the Pin Map Editor displays the aliased resource names, rather than the model description resource names.

If present, the system description file is located in the following path: `<Program Files>\National Instruments\Shared\InstrumentLibrary\SystemDescriptions\SystemDescription.xml`. You can define your own system description file and copy it to `<Public>\Documents\National Instruments\InstrumentLibrary\SystemDescriptions\SystemDescription.xml`.

Adding a Model-Based Instrument to Your Pin Map (TSM)

Use the Instruments tab of the Pin Map Editor to add and modify instances of Model-Based Instruments in the pin map.

The Pin Map Editor reads the model description files for all Model-Based Instruments in the pin map. If you modify an instrument model file, you must exit and relaunch the Pin Map Editor to see the changes in the Pin Map.

1. Navigate to **Semiconductor Module»Edit Pin Map File** to launch the Pin Map Editor.
2. In the Pin Map tab, select **<Add Instruments Here>**.
   TSM displays the instrument types that you can add.
3. Click **Model-Based Instrument** to launch the configuration window.
4. In the **Name** field, specify the name of the instrument that will appear in the pin map.
5. Select the instrument model description file from the **Instrument Model** drop-down menu.
   TSM will display the category and subcategory of the Model-Based Instrument you select.
6. Specify any additional instrument or resource properties.
7. Click **OK**.

Pin Map File XML Structure (TSM)

The pin map XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\PinMap.xsd`, defines the following structure for a pin map XML file:

Legend	
R	**<Root element>**
E	**<Element>**
A	`Attribute`

<PinMap>

- schemaVersion—Specifies the version of the schema file.
- **<Instruments>**—Specifies the type of instruments connected to the tester, the name of each instrument, and the number of channels available for each instrument.

Note Consider using the following instrument naming convention for semiconductor test programs: `InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation`, for example, `HSD_657x_C2_S03`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), and `SlotLocation` uses double digits to identify the slot location (`Sxx`).

- <NIDigitalPatternInstrument>—Defines an NI-Digital Pattern instrument.
  - name—Name of the instrument, as defined in Measurement & Automation Explorer (MAX).
  - numberOfChannels—Number of channels available on the instrument.
  - group—Name of the group that contains the instrument. Group names are case sensitive. By default, the Pin Map Editor sets this attribute to `Digital` when you add NI-Digital Pattern instruments to the pin map file. By using the same group name for all NI-Digital Pattern instruments, TSM combines all instruments into a single session so you can avoid session loops in code modules. To create multiple NI-Digital Pattern sessions, use a unique name for each set of instruments for which you want to create a session. Refer to the **Digital Pattern Help** for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

- E **<NIDCPowerInstrument>**—Defines an NI-DCPower instrument.
  - A `name`—Name of the instrument, as defined in MAX.
  - A `numberOfChannels`—Number of channels available on the instrument.
  - E **<ChannelGroup>**—Defines a group of channels controlled by one session. By grouping channels into a single session, you can avoid using session loops in code modules. By default, the Pin Map Editor creates one channel group containing all instrument channels. To create multiple, custom groups, use a unique name for the set of instrument channels for which you want to create a session. Note that channels within a group do not have to be from the same NI-DCPower instrument. Refer to the NI-DCPower Help for information about independent channels.
    - A `name`—Name of a group of channels. Group names are case sensitive.
    - A `channels`—Channel(s) that are assigned to a group. If not defined, TSM will assign all channels from the instrument. Channels can be defined as a comma-separated list (e.g., 0,1,3,..,n), a continuous range (e.g., 0:3), or as a combination of the two (e.g., 0:1,3). All channels from an instrument must be assigned to a group and a channel cannot be in multiple groups.
- E **<NIDAQmxTask>**—Defines an NI-DAQmx task, not an instrument.
  - A `name`—Name of the task, as defined in test program code modules.
  - A `taskType`—Category of the task. Pin queries that return tasks of more than one task type return an error.
  - A `channelList`—List of physical channels associated with the task.
- E **<NIDmmInstrument>**—Defines an NI-DMM instrument. NI-DMM instruments define a single channel, displayed within TSM as channel 0.
  - A `name`—Name of the instrument, as defined in MAX.
- E **<NIFGenInstrument>**—Defines an NI-FGEN instrument.

- A `name`—Name of the instrument, as defined in MAX.
- A `numberOfChannels`—Number of channels available on the instrument.

- E **<NIScopeInstrument>**—Defines an NI-SCOPE instrument.
  - A `name`—Name of the instrument, as defined in MAX.
  - A `numberOfChannels`—Number of channels available on the instrument.
  - A `group`—Name of the group that contains the instrument. Group names are case sensitive. By default, the Pin Map Editor sets this attribute to `Scope` when you add NI-SCOPE instruments to the pin map file. Group names are case sensitive. By using the same group name for all NI-SCOPE instruments, TSM combines all instruments into a single session so you can avoid session loops in code modules. To create multiple NI-SCOPE sessions, use a unique name for each set of instruments for which you want to create a session. Refer to the **NI-SCOPE Help** for information about hardware limitations that prevent certain instruments from operating together as a single instrument.
- E **<NIHSDIOInstrument>**—Defines an NI-HSDIO instrument.
  - A `name`—Name of the instrument, as defined in MAX.
  - A `numberOfChannels`—Number of channels available on the instrument.
  - A `PFILines`—(Optional) Defines the PFI lines available in the NI-HSDIO instrument in a comma-separated list of numbers or ranges of numbers separated by a hyphen. PFI number ranges are inclusive and must be in ascending order. Example: `PFILines=2,3,4-8`
- E **<NIRFSAInstrument>**—Defines an NI-RFSA instrument. NI-RFSA instruments define a channel named `In`.
  - A `name`—Name of the instrument, as defined in MAX.
- E **<NIRFSGInstrument>**—Defines an NI-RFSG instrument. NI-RFSG instruments define a channel named `Out`.

- A `name`—Name of the instrument, as defined in MAX.

- E **<NIVSTInstrument>**—Defines an NI-VST instrument that can hold RFSA, RFSG, and FPGA sessions. NI-VST instruments define a channel named `In` and another channel named `Out`.
  - A `name`—Name of the instrument, as defined in MAX.
  - A `fpgaFilePath`—(Optional) Path to the FPGA file relative to the path of the pin map file. You can manually specify an absolute file path.

- E **<NIRFPMInstrument>**—Defines an RF Port Module instrument that can hold RFPM, RFmx, RFSA, RFSG, and FPGA sessions.
  - A `name`—Name of the VST instrument, as defined in MAX, that is part of the RF port module subsystem.
  - A `portsList`—Defines the ports available in the RF Port Module in a comma-separated list of numbers or ranges of numbers separated by a hyphen. Port number ranges are inclusive and must be in ascending order, for example, `channelList="2,3,4-8"`.
  - A `calibrationFilePath`—Path, relative to the path of the pin map file, to the TDMS files that contain the calibration data for the RF Port Module instrument. You can manually specify an absolute file path.
  - A `iviSwitchName`—IVI Switch resource name associated with the port module, as defined in MAX.
  - A `fpgaFilePath`—(Optional) Path to the FPGA file relative to the path of the pin map file. You can manually specify an absolute file path.

- E **<NI5530RFPortModule>**—Defines an NI-5530 RF Port Module instrument. You can use the NI-5530 RF Port Module to multiplex one RF instrument across multiple test sites or multiple RF instruments across multiple test sites. This element is replaced by the NIRFPMInstrument.
  - A `name`—Name of the instrument, as defined in MAX.
  - A `calibrationFilePath`—Path, relative to the path of the pin map file, to the TDMS files that contain the calibration data for the RF Port Module instrument. You can manually specify an absolute file path.

- E **<NIRelayDriverModule>**—Defines a PXI-2567 relay driver module.

- A `name`—Name of the relay driver module, as defined in MAX.
- A `numberOfControlLines`—Number of control lines available on the relay driver module.

- E **<Instrument>**—Defines an instrument that TSM does not natively support. Use the TSM Code Module API to set any type of session data on a channel, group of channels, or instrument. Refer to the example sequence files, pin map files, and LabVIEW code module VIs located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments` directory for examples of using TSM pin map files and VIs to perform tests using instruments that TSM does not natively support.
  - A `name`—String that identifies the instrument. For instruments that NI provides but that TSM does not natively support, specify the name of the instrument, as defined in MAX.
  - A `instrumentTypeId`—String that identifies the instrument type, family, class, or product group. You cannot specify a value that begins with `ni`. This value is a string that you define in the pin map and is not a predefined value from some other source, such as a name in MAX, that you select. Use this value to identify all instances of a particular instrument type. Instruments of the same type typically have the same session data type and same driver API.
  - E **<ChannelGroup>**—Defines a synchronized group of channels. Specify individual `<Channel>` elements with unique IDs within the channel group.
    - A `id`—Unique ID for the channel group. An instrument cannot contain more than one channel group with the same ID.
    - E **<Channel>**—Channel within the channel group.
      - A `id`—Unique ID for the channel. An instrument cannot contain more than one channel with the same ID.
  - E **<Channel>**—Channel on the instrument.
    - A `id`—Unique ID for the channel. An instrument cannot contain more than one channel with the same ID.

- E **<NIModelBasedInstrument>**—Defines a Model-Based Instrument.
  - A `name`—Unique string that identifies the instance of the Model-Based Instrument in the pin map.
  - A `instrumentModel`—Installed model description files in the model library.
  - A `category`—String that specifies the category to which the instrument belongs. The instrument model description defines the category for the instrument model.
  - A `subcategory`—(Optional) String that specifies the subcategory to which the instrument belongs. The instrument model description defines the subcategory for the instrument model.
  - E **<Resource>**—(Optional) If required by the model description file, specifies the instrument resource name in Measurement & Automation Explorer (MAX).
    - A `owner`—Specifies the instrument resource in the model description file to which the attribute values of the `<UserData>` element apply.
    - E **<UserData>**—Contains the properties from the model description file you can assign.
      - A `propertyName`—Name of the property defined in the model description file.
      - A `propertyValue`—Value you assign to the property.
- E **<Multiplexer>**—Defines a switching instrument to use as a multiplexer across multiple test sites. You can use one instrument multiplexed across multiple test sites or multiple instruments multiplexed across multiple test sites.
  - A `name`—Name of the Switch Executive virtual device, as defined in MAX.
  - A `multiplexerTypeId`—(Optional) String that identifies the switch type, family, class, or product group. You cannot specify a value that begins with `ni`. This value is a string that you define in the pin map

and is not a predefined value from some other source, such as a name in MAX, that you select. Use this value to identify all instances of a particular switch type. Switches of the same type typically have the same session data type and same driver API.

- E **<Pins>**—Specifies the pins on the DUT and the pins on the tester that the test program associated with the pin map file references.
  - E **<DUTPin>**—Defines a DUT pin, which is a pin on a DUT or a resource on the tester or DIB that is associated with one or more sites.
    - A `name`—String that identifies the DUT pin.
  - E **<SystemPin>**—Defines a system pin, which is resource on the tester or DIB that is connected to an instrument.
    - A `name`—String that identifies the system pin.
- E **<PinGroups>**—Specifies named grouping of pins.
  - E **<PinGroup>**—Defines a group of pins that you can reference with a single name.
    - A `name`—String that identifies the group of pins.
    - E **<PinReference>**—Specifies a pin or a group of pins within the pin group.
      - A `pin`—String that specifies the name of an existing pin or pin group.
- E **<Relays>**—Specifies the relays on the site and the relays on the tester that the test program associated with the pin map file references.
  - E **<SiteRelay>**—Defines a site relay, which is a relay on the tester or DIB that is connected to a relay driver module and that is associated with one or more sites.
    - A `name`—String that identifies the site relay.
    - A `openStateDisplayLabel`—(Optional) A description of the connections when the relay is in the open state. This attribute is only for informational and display purposes for the Digital Pattern Editor.

- A `closedStateDisplayLabel`—(Optional) A description of the connections when the relay is in the closed state. This attribute is only for informational and display purposes for the Digital Pattern Editor.

- E **<SystemRelay>**—Defines a system relay, which is a relay on the tester or DIB that is connected to a relay driver module and that is associated with all sites.
  - A `name`—String that identifies the system relay.
  - A `openStateDisplayLabel`—(Optional) A description of the connections when the relay is in the open state. This attribute is only for informational and display purposes for the Digital Pattern Editor.
  - A `closedStateDisplayLabel`—(Optional) A description of the connections when the relay is in the closed state. This attribute is only for informational and display purposes for the Digital Pattern Editor.

- E **<RelayGroups>**—Specifies named grouping of relays.
  - E **<RelayGroup>**—Defines a group of relays that you can reference with a single name.
    - A `name`—String that identifies the group of relays.
    - E **<RelayReference>**—Specifies a relay or a group of relays within the relay group.
      - A `relay`—String that specifies the name of an existing relay or relay group.

- E **<RelayConfigurations>**—Specifies a grouping of relay configurations.
  - E **<RelayConfiguration>**—Defines a relay configuration. A relay configuration is the name assigned to a set of relays and their positions.
    - A `name`—String that identifies the relay configuration.
    - E **<RelayPosition>**—Specifies a relay and its position.
      - A `relay`—String that specifies the name of an existing relay or relay group.

- A position—String that specifies the position of the relay or relay group. Valid values are Open or Closed.

- E **<Sites>**—Specifies the sites on the tester.
  - E <Site>—Defines a site to test.
    - A siteNumber—Number that identifies the site. Site numbers must start at 0 and be consecutive without gaps.
- E **<Connections>**—Specifies mappings among pins, sites, instruments, and instrument channels.
  - E **<Connection>**—Defines a connection between a DUT pin and an instrument channel.
    - A pin—Name of the DUT pin to connect. The value must match the value of the name attribute of a <DUTPin> element.
    - A siteNumber—The site or group of sites associated with the connection. The value must match the value of the siteNumber attribute of one of the <Site> elements or it must be a comma-separated list of site numbers.
    - A instrument—Name of the instrument or DAQmx task to connect. The value must match the value of the name attribute of an <Instrument> element.
    - A channel—ID of the instrument channel or physical channel ID of the DAQmx task to connect.
    - A deembeddingFilePath—Path, relative to the path of the pin map file, to the S2P file for de-embedding an RF Port Module connection. You can manually specify an absolute path.
    - A deembeddingOrientation—(Optional) Used with the deembeddingFilePath attribute to specify the orientation of the data in the S2P file relative to the port the channel attribute specifies. Valid values are **Port1TowardDUT** or **Port2TowardDUT**.
  - E **<SystemConnection>**—Defines a direct connection between a system pin and an instrument channel.

- A `pin`—Name of the system pin to connect. The value must match the value of the `name` attribute of a `<SystemPin>` element.
- A `instrument`—Name of the instrument to connect. The value must match the value of the `name` attribute of an `<Instrument>` element.
- A `channel`—(Optional) ID of the instrument channel to connect.
- A `deembeddingFilePath`—Path, relative to the path of the pin map file, to the S2P file for de-embedding an RF Port Module connection. You can manually specify an absolute path.
- A `deembeddingOrientation`—(Optional) Used with the `deembeddingFilePath` attribute to specify the orientation of the data in the S2P file relative to the port the `channel` attribute specifies. Valid values are **Port1TowardDUT** or **Port2TowardDUT**.

- E **<MultiplexedConnection>**—Defines a multiplexed connection between the same DUT pin on multiple sites and a single instrument channel.
  - A `instrument`—Name of the instrument to connect. The value must match the value of the `name` attribute of an `<Instrument>` element.
  - A `channel`—ID of the instrument channel to connect.
  - E **<MultiplexedDUTPinRoute>**—Specifies the route required to connect a DUT pin on a specific site to the instrument channel.
    - A `pin`—Name of the DUT pin to connect. The value must match the value of the `name` attribute of a `<DUTPin>` element.
    - A `siteNumber`—Site for the DUT pin in the system. The value must match the value of the `siteNumber` attribute of a `<Site>` element.
    - A `multiplexer`—String that identifies the multiplexer required to create the route. The value must match the value of the `name` attribute of a `<Multiplexer>` element.
    - A `routeName`—String that identifies the multiplexer route required to connect the pin and site to the instrument and channel.

- A deembeddingFilePath—Path, relative to the path of the pin map file, to the S2P file for de-embedding an RF Port Module connection. You can manually specify an absolute path.
- A deembeddingOrientation—(Optional) Used with the deembeddingFilePath attribute to specify the orientation of the data in the S2P file relative to the port the channel attribute specifies. Valid values are **Port1TowardDUT** or **Port2TowardDUT**.

- E **<RelayConnection>**—Defines a connection between a site relay and a control line of a relay driver module.
  - A relay—Name of the site relay to connect. The value must match the value of the name attribute of a <SiteRelay> element.
  - A siteNumber—The site or group of sites associated with the connection. The value must match the value of the siteNumber attribute of one of the <Site> elements or it must be a comma-separated list of site numbers.
  - A relayDriverModule—Name of the relay driver module to connect. The value must match the value of the name attribute of an <NIRelayDriverModule> element.
  - A controlLine—ID of the physical control line of the relay driver module to connect.
- E **<SystemRelayConnection>**—Defines a direct connection between a system relay and a control line of a relay driver module.
  - A relay—Name of the system relay to connect. The value must match the value of the name attribute of a <SystemRelay> element.
  - A relayDriverModule—Name of the relay driver module to connect. The value must match the value of the name attribute of an <NIRelayDriverModule> element.
  - A controlLine—ID of the physical control line of the relay driver module to connect.

Common XML Validation Error Messages (TSM)

If the contents of the pin map XML file do not satisfy the constraints the pin map schema defines, TSM reports error messages. Some of the error messages are generic XML validation errors and can be difficult to decipher. Refer to the following tables to interpret certain error messages.

Error Message

```
The key sequence '<Item>' in 'http://www.ni.com/TestStand/
SemiconductorModule/PinMap.xsd:<Element>' Keyref fails to
refer to some key.
```

Meaning

The following table includes the exact meaning of the error message, which depends on the value of the `<Element>` text.

Value of `<Element>`	Meaning of Error Message
PinName	The specified pin `<Item>` does not exist in the pin map.
SystemPinName	The specified system pin `<Item>` does not exist in the pin map.
PinOrPinGroupName	The specified pin or pin group name `<Item>` does not exist in the pin map.
RelayName	The specified relay `<Item>` does not exist in the pin map.
SystemRelayName	The specified system relay `<Item>` does not exist in the pin map.
RelayOrRelayGroupName	The specified relay or relay group name `<Item>` does not exist in the pin map.
SiteNumber	The specified site number `<Item>` does not exist in the pin map.
InstrumentName	The specified instrument `<Item>` does not exist in the pin map.
MultiplexerName	The specified multiplexer `<Item>` does not exist in the pin map.

RelayDriverModuleName	The specified relay driver module <Item> does not exist in the pin map.

Error Message

```
There is a duplicate key sequence '<Item1> [<Item2>]' for
the 'http://www.ni.com/TestStand/SemiconductorModule/PinMa
p.xsd:<Element>' key or unique identity constraint.
```

Meaning

The following table includes the exact meaning of the error message, which depends on the value of the <Element> text.

Value of <Element>	Meaning of Error Message
PinName, AllPinNames, AllPinOrPinGroupNames, AllPinAndRelayNames	The specified pin, pin group name, relay, or relay group name <Item1> is defined multiple times in the pin map.
SiteNumber	The specified site number <Item1> is defined multiple times the pin map.
InstrumentName	The specified instrument <Item1> is defined multiple times the pin map.
MultiplexerName	The specified multiplexer <Item1> is defined multiple times in the pin map.
RelayDriverModuleName	The specified relay driver module <Item1> is defined multiple times in the pin map.
ConnectionDUTPin	The pin <Item2> on site <Item1> is connected to multiple instrument channels.
ConnectionSiteRelay	The relay <Item2> on site <Item1> is connected to multiple relay driver module control lines.
SystemConnectionDUTPin	The system pin <Item1> is connected to multiple instrument channels.
SystemConnectionSiteRelay	The system relay <Item1> is connected to multiple relay driver module control lines.
ConnectionInstrumentChannel	The channel <Item2> on instrument <Item1> is connected to multiple pins.

RelayConnectionModuleDriver	The control line <Item2> on relay driver module <Item1> is connected to multiple relays.
MultiplexedRouteName	The multiplexer route <Item1> on multiplexer <Item2> has duplicate connections.
UniqueChannelAndChannelGroup	The specified channel or channel group name <Item1> is defined multiple times in the pin map.

Schema Version Policy (TSM)

The schema version uses a **major.minor** notation. The version of the schema reflects changes to the schema, not changes to TSM.

Changes to the schema version indicate breaking changes to the schema. TSM does not load files that use a later schema version than the version specified for that version of TSM.

Use the following table to map schema versions and TSM versions.

Schema Version	TSM Version
PinMap.xsd version 1.0 BinDefinitions.xsd version 1.0	NI TestStand 2013 Semiconductor Module
PinMap.xsd version 1.1 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module
PinMap.xsd version 1.2 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module SP1 NI TestStand 2016 Semiconductor Module
PinMap.xsd version 1.3 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2016 SP1 Semiconductor Module
PinMap.xsd version 1.4 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2017 Semiconductor Module
PinMap.xsd version 1.5 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.0	NI TestStand 2019 Semiconductor Module

PinMap.xsd version 1.6 BinDefinitions.xsd version 1.2 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.1	NI TestStand 2020 Semiconductor Module TestStand 2020 Semiconductor Module 2021 Q4

Connecting Shared Resources in the Pin Map (TSM)

A shared resource is a device on the tester or DIB that is connected to an instrument or relay driver module and shared by multiple sites. To specify a connection between a shared resource and an instrument channel or relay driver module control line, use the following criteria to decide how to define a shared resource in the pin map.

1. If the resource is a relay:
    a. If there is a single relay shared by all sites, define a system relay.
    b. If there are multiple relays in which each relay is associated with multiple sites, define a site relay.
2. If the resource is not a relay:
    a. If the resource is shared by all sites and you do not need to burst patterns to the resource using an NI-Digital Pattern instrument, define a system pin.
    b. Otherwise, define a DUT pin.

Connecting a Shared Resource using System Pins and System Relays

For each system pin and system relay in the pin map, the Pin Map Editor displays a row in the Connections table. Complete the following steps in the Pin Map Editor to create a connection for a system pin or system relay that all sites share.

1. Select **Connections** on the Pin Map tab.
2. In the **View Connections for** drop-down menu, select **All Pins and Relays**.
3. In the Connections table, select the instrument and channel or relay driver module and control line from the Instrument and Channel column cells of the system pin or system relay for which you want to create a connection.

Connecting a Shared Resource to One or More Sites using DUT Pins and Site Relays

For each DUT pin and site relay, the Pin Map Editor assumes there is one connection per site and displays a row for each site in the Connections table. Complete the following steps in the Pin Map Editor to create a connection for a DUT Pin or site relay that multiple sites share.

1. Select **Connections** on the Pin Map tab.
2. In the **View Connections for** drop-down menu, select **All Pins and Relays**.
3. In the Connections table, select the instrument and channel or relay driver module and control line from the Instrument and Channel column cells for one of the rows associated with the DUT pin or site relay.
4. Repeat step 3 for the remaining sites, using the same instrument channel or relay driver module control line for the sites that share the resource. The Pin Map Editor automatically combines the rows that use the same resource into a single row. Alternatively, you can enter a comma-separated list of site numbers in the Site column to specify which sites share the resource.

When you save the pin map file, the Pin Map Editor automatically removes any duplicate site connections from the Connections table.

When you define a shared resource in the pin map, the results of your Pin Query VI or .NET method and Publish Data VI or `Publish` .NET method will be affected as follows:

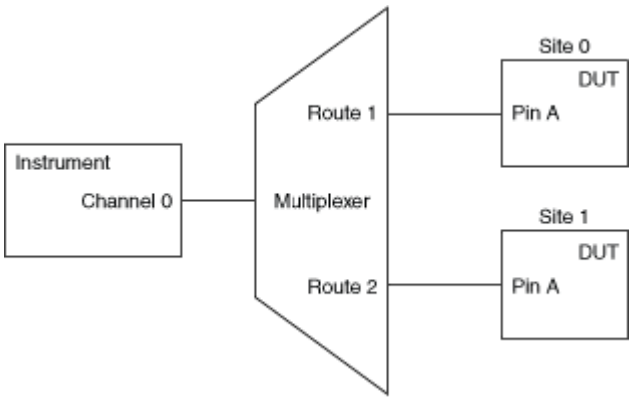
- **Pin Query VI or .NET method**—Only one entry is included for a shared resource.
- **Publish Data VI or `Publish` .NET method**—The result is duplicated for each site that uses the shared resource.

Specifying Multiplexers and Multiplexed Connections in a Pin Map (TSM)

You must define the switch and route in the pin map for the test program to access a switch route from the TSM Code Module API. Refer to the `Switching.pinmap` in

the Switching example for an example of a pin map that contains multiplexed connections.

The pin map specifies switches as <Instruments>/<Multiplexer> elements. Each connection between an instrument channel and a DUT pin that is routed through a switch must specify a <Connections>/<MultiplexedConnection> element, as shown in the following figure.



The <MultiplexedConnection> element defines a connection between a single instrument channel and the same DUT pin on multiple sites. The <MultiplexedConnection> element also contains a list of <MultiplexedDUTPinRoute> elements that each specify the route required to connect an instrument channel to a DUT pin on a specific site.

Binning DUTs Based on Test Results (TSM)

A handler moves each tested device under test (DUT) from its test site to an appropriate physical bin (hardware bin), depending on the test results. The tester software assigns each tested DUT to a hardware bin and communicates with the handler to ensure that the handler places the DUT in the appropriate hardware bin.

Typically, the handler places passed DUTs in one or more hardware bins, depending on the grade of the DUT, and places failed DUTs in multiple hardware bins, depending on the type or condition of failure.

Because handlers have a limited quantity of hardware bins available, the tester software also assigns each DUT to a virtual bin defined in software (software bin) to provide a more detailed classification of test results. The tester software records the software bin for each DUT in a Standard Test Data Format (STDF) log file for

additional analysis after testing completes. When wafer testing, the prober assigns hardware bins to individual die based on the test results. Because probers generally do not have the same limitations as handlers on the number of hardware bins they support, the set of software bins can be identical to the set of hardware bins when testing wafers. Use the Bin Definitions Editor and enable the **Software Bins Only** option to ensure that software bins and hardware bins are the same.

TSM Implementation

Use the Bin Definitions Editor to define the software bins and hardware bins for the test program, define how the software bins relate to hardware bins, and define the default software bins in the test program. The TSM installs a bin definitions XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\BinDefinitions.xsd`, which you can use to create a valid bin definitions file. Use the **Bin Definitions File Path** control on the Bin Definitions panel of the Test Program Editor to specify the bin definitions file to use with the test program.

The test program refers only to software bins. Because each software bin is associated with a hardware bin, the tester assigns a hardware bin to a DUT when it assigns the associated software bin to the DUT.

Each instance of the Semiconductor Multi Test step contains one or more tests for which you can specify the software bin to assign to the DUT when a test fails. When you specify a valid bin definitions file on the Bin Definitions panel of the Test Program Editor, you can use the **Software Bin** column on the Tests tab to select a software bin defined in the bin definitions file.

Note The first Semiconductor Multi Test step test failure determines the software bin for the DUT. Once a software bin has been assigned to the DUT, subsequent Semiconductor Multi Test step test failures do not change the bin assignment.

In addition, you can use a Set and Lock Bin step to arbitrarily assign a software bin to a DUT. Use the **Bin Expression** control on the Set and Lock Bin tab to specify an expression that evaluates at run time to a valid software bin number defined in the bin definitions file. The step assigns a software bin to the DUT and locks the bin by

preventing tests on subsequent Semiconductor Multi Test steps from assigning a bin to the DUT when the test fails. You can use the Set and Lock Bin step to implement grading in the test program.

See Also

Reports and Data Logs

Retesting DUTs

Bin Definitions File XML Structure (TSM)

The bin definitions XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\BinDefinitions.xsd`, defines the following structure for a bin definitions XML file:

Legend	
R	**<Root element>**
E	**<Element>**
A	`Attribute`

R <BinDefinitions>

- A `schemaVersion`—Specifies the version of the schema file.
- A `softwareBinsOnlyMode`—(Optional) Specifies whether the bin definitions editor displays only software bins and no hardware bins. The bin definitions editor sets this attribute when you enable the **Software Bins Only** option. This option can be useful when you perform wafer testing using software bins exclusively. When this attribute is `True`, the bin definitions editor ensures that a unique hardware bin exists for each software bin.
- E **<HardwareBins>**—Specifies the hardware bins available to the handler or prober.
  - E **<Bin>**—Specifies an individual hardware bin.
    - A `name`—(Optional) A short descriptive name for the hardware bin. TSM stores the name in the Hardware Bin Record (HBR) of a Standard Test Data Format (STDF) log file.

- A `number`—A unique number the handler or prober uses to place the device under test (DUT). The number must be a valid 16-bit unsigned integer.
- A `type`—Specifies a value of `Pass`, `Fail`, or `Other`.

- E **<SoftwareBins>**—Specifies the software bins to define for the test program.
  - A `errorBin`—The software bin number TSM assigns to a DUT when the main test sequence errors. The software bin must be associated with a hardware bin of the `Fail` or `Other` type.
  - A `defaultFailBin`—(Optional) The software bin number TSM assigns to a DUT when the main test sequence fails and a bin has not yet been assigned to the DUT. The software bin must be associated with a hardware bin of the `Fail` or `Other` type. You can omit the default fail bin from the bin definitions file. In this case, TSM uses the `errorBin` for the `defaultFailBin`.
  - A `defaultPassBin`—The software bin number TSM assigns to a DUT when the main test sequence passes and a bin has not yet been assigned to the DUT. The software bin must be associated with a hardware bin of the `Pass` type.
  - E **<Bin>**—Specifies an individual software bin.
    - A `name`—(Optional) A short descriptive name for the software bin. The Semiconductor Multi Test step displays this name in the **Software Bin** column on the Tests tab when you select the software bin for a DUT. TSM stores the name in the Software Bin Record (SBR) of an STDF log file.
    - A `number`—A unique number to identify the software bin. The number must be a valid 16-bit unsigned integer.
    - A `hardwareBin`—The hardware bin number to associate with the software bin. The type of the software bin is inferred from the associated hardware bin. The Software Bin column on the Tests tab of the Semiconductor Multi Test step lists only software fail bins. TSM ensures that the test status of a DUT corresponds to the type of

software bin assigned to the DUT. For example, TSM reports a run-time error if a test program causes TSM to assign a software pass bin to a failed DUT.

See Also

Bin Definitions Editor

Reports and Data Logs

Schema Version Policy (TSM)

The schema version uses a **major.minor** notation. The version of the schema reflects changes to the schema, not changes to TSM.

Changes to the schema version indicate breaking changes to the schema. TSM does not load files that use a later schema version than the version specified for that version of TSM.

Use the following table to map schema versions and TSM versions.

Schema Version	TSM Version
PinMap.xsd version 1.0 BinDefinitions.xsd version 1.0	NI TestStand 2013 Semiconductor Module
PinMap.xsd version 1.1 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module
PinMap.xsd version 1.2 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module SP1 NI TestStand 2016 Semiconductor Module
PinMap.xsd version 1.3 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2016 SP1 Semiconductor Module
PinMap.xsd version 1.4 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2017 Semiconductor Module
PinMap.xsd version 1.5 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.0	NI TestStand 2019 Semiconductor Module

PinMap.xsd version 1.6 BinDefinitions.xsd version 1.2 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.1	NI TestStand 2020 Semiconductor Module TestStand 2020 Semiconductor Module 2021 Q4

Grading Passed DUTs (TSM)

You can use a Set and Lock Bin step to grade a passed DUT, in which the test program evaluates the DUT with different test criteria and assigns a pass bin to the DUT depending on the level of criteria the DUT met.

Complete the following steps to implement grading in a TSM test program.

1. Set the default pass bin in the bin definitions file to the software bin associated with the lowest passing grade.
2. Insert a Semiconductor Multi Test step in the sequence and complete the following steps to configure the step to test the highest grade.
    a. On the Module tab of the Step Settings pane, specify a code module that takes a measurement to use for determining the DUT grade.
    b. On the Tests tab, add a single test and use the **Export Measurement To** column to specify a local variable to use for storing the measurement value, which you will use in subsequent steps.
    c. Use the **Low Limit** and **High Limit** columns to specify the limit set that determines the highest passing grade.
    d. Leave the **Software Bin** column empty because a failure on a high-grade test must not assign the DUT to a fail bin.
    e. Disable the **Step Failure Causes Sequence Failure** option on the Run Options panel of the Properties tab.
3. Insert another Semiconductor Multi Test step and complete the following steps to configure the step to test the next lower grade.
    a. On the Preconditions panel of the Properties tab, specify a precondition expression so that the step executes only if the step for the next higher grade fails. TSM does not need to perform this test if the DUT already passed a higher grade test.

b. On the Tests tab, add a single test and use the **Test Data Source** column to specify the local variable in which you stored the measurement value in step 2b.

c. Use the **Low Limit** and **High Limit** columns to specify the limit set that determines the passing grade.

d. If this step does not test the lowest passing grade, leave the **Software Bin** column empty. Otherwise, specify the software bin to assign if the DUT fails the lowest passing grade.

e. If this step does not test the lowest passing grade, disable the **Step Failure Causes Sequence Failure** option on the Run Options panel. If the step tests the lowest passing grade, enable the **Step Failure Causes Sequence Failure** option.

4. Repeat step 3 to add additional Semiconductor Multi Test steps and configure the steps to test each of the next lower grades until the lowest passing grade.
5. At the end of the sequence, insert a Set and Lock Bin step for each grade, starting from the second lowest to the highest, and complete the following steps to configure each Set and Lock Bin step.

   a. On the Set and Lock Bin tab, use the **Bin Expression** control to specify the software pass bin you want to associate with the grade.

   b. On the Preconditions panel of the Properties tab, specify a precondition expression so that the step executes only if the sequence passed (`!RunState.SequenceFailed`) and all the tests for the grade passed.

See Also

Grading Example

Specifications Files (TSM)

Use an XML-based specifications file (`.specs`) to define a set of variables and associated numeric values that you can reference in test program code modules instead of using constants to set testing specifications. You can modify specifications files to set new values without changing test program files. You can

specify the values directly or calculate the values from other variables using formulas you write with simple arithmetic expressions.

Select **Semiconductor Module»Edit Test Program** and select **Specifications Files** in the Test Program Editor to launch the Specifications Files panel. Use the **Specifications File Path** control on the Specifications Files panel to specify one or more specifications files to use with the test program.

Note You can use the Digital Pattern Editor (if installed) to view, create, modify, and save specifications files. Any changes to specification values you make using the NI-Digital Pattern Driver API do not affect the value of the specifications the TSM Code Module API returns.

Use the Get Specification(s) Value(s) TSM Code Module API VI or the `GetSpecificationsValue` or `GetSpecificationsValues` TSM Code Module API .NET methods to query one variable or an array of variables and return the calculated value or values. The VI and methods return an error or exception when you reference a variable that does not exist in the specifications file, which can result in a run-time error when the sequence executes.

The specifications XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\Specifications.xsd`, defines the structure for a specifications XML file.

Configuring Specifications

Each variable must include a section name you designate by the text you type before the period (.) in the Section.Variable column. For example, the section name for the variable `DC.vcc` is `DC`. Variables must be unique within each section of a specifications file and among all specifications files associated with the test program, including specifications files defined in the digital pattern project you associate with the test program.

You can specify the following components of each specification:

- **Section.Variable**—Name of the variable to use when querying the value in test code. Variable names are case sensitive and must begin with a letter or

underscore (_), are limited to A–Z, a–z, 0–9, or _ characters, and must include a period (.) to separate the section name from the variable name. Do not use math function names as variable names.

- **Formula** —Numeric value or formula definition for the variable. When you reference other variables to calculate a value, ensure that you use consistent units. Use the following options to specify a formula definition:

Option	Valid Values	Example(s)
Numeric values	—	5.0 -115.003 3.4E-15 150 ns 3.4 mV
Simple math operators	+, –, /, *, ( )	5.0 + (3.14 * 2)
Math functions	abs, ceil, floor, fmod, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, deg, rad, pi, exp, pow, log, log10, sqrt, min, max	abs(-15) * 2
Variable references	—	AC.Variable1 * 15 where Variable1 is another specification within the AC section
Units	**Units: (take precedence over SI prefixes)** ▪ V, volts, Volts, volt, Volt ▪ A, amps, Amps, amp, Amp, amperes, Amperes, ampere, Ampere ▪ s, sec, Sec, secs, Secs, second, Second, seconds, Seconds	10 ns + 15 ns

- dB, db, decibel, Decibel, decibels, Decibels
- dBm, dbm
- F, farad, Farad, farads, Farads
- H, henries, Henries, henry, Henry
- Hz, hertz, Hertz
- Ω, ohms, Ohms, ohm, Ohm
- W, watts, Watts, watt, Watt

SI Prefixes:

- d, deci (scaling factor of 1E-1)
- c, centi (scaling factor of 1E-2)
- m, milli (scaling factor of 1E-3)
- µ, u, micro (scaling factor of 1E-6)
- n, nano (scaling factor of 1E-9)
- p, pico (scaling factor of 1E-12)
- f, femto (scaling factor of 1E-15)
- da, deca (scaling factor of 1E+1)
- h, hecto (scaling factor of 1E+2)
- k, kilo (scaling factor of 1E+3)
- M, mega (scaling factor of 1E+6)

	▪ G, giga (scaling factor of 1E+9) ▪ T, tera (scaling factor of 1 E+12) ▪ P, peta (scaling factor of 1E+15)	

▪ **Comment**—Describes what the formula represents. TSM does not display the description.

Specifications File XML Structure (TSM)

The specifications XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\Specifications.xsd`, defines the following structure for a specifications XML file.

Legend	
R	**<Root element>**
E	**<Element>**
A	`Attribute`

R <Specifications>

▪ A `schemaVersion`—Specifies the version of the schema file.

▪ E **<Section>**—You can use multiple `<Section>` elements in a specifications file.

▪ A `name`—Name of the section of the specifications file.

▪ E **<f:Formula>**—Defines and describes an association of a symbol to its value specification.

▪ A `symbol`—Name of the formula to use when querying the value in test code. Symbol names must begin with a letter or underscore (_) and are limited to A-Z, a-z, 0-9, or _ characters. Symbol names are case sensitive. Do not use math function names as symbol names.

▪ E **<f:Definition>**—Value specification for the formula that TSM interprets to calculate a numeric value when you query the symbol.

When you reference other formulas to calculate a value, ensure that you use consistent units.

- **<f:Description>**—Describes what the formula represents. TSM does not display the description.

Schema Version Policy (TSM)

The schema version uses a **major.minor** notation. The version of the schema reflects changes to the schema, not changes to TSM.

Changes to the schema version indicate breaking changes to the schema. TSM does not load files that use a later schema version than the version specified for that version of TSM.

Use the following table to map schema versions and TSM versions.

Schema Version	TSM Version
PinMap.xsd version 1.0 BinDefinitions.xsd version 1.0	NI TestStand 2013 Semiconductor Module
PinMap.xsd version 1.1 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module
PinMap.xsd version 1.2 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module SP1 NI TestStand 2016 Semiconductor Module
PinMap.xsd version 1.3 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2016 SP1 Semiconductor Module
PinMap.xsd version 1.4 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2017 Semiconductor Module
PinMap.xsd version 1.5 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.0	NI TestStand 2019 Semiconductor Module
PinMap.xsd version 1.6 BinDefinitions.xsd version 1.2 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.1	NI TestStand 2020 Semiconductor Module TestStand 2020 Semiconductor Module 2021 Q4

Digital Patterns (TSM)

Digital pattern files contain a collection of vectors, or instructions, to execute on an NI-Digital Pattern instrument. Components of the binary pattern file include time sets, labels, opcodes, vector numbers, pin state data that indicates drives and compares, and comments for each vector. You can edit pattern files in the Digital Pattern Editor.

TSM Implementation

Select **Semiconductor Module»Launch Digital Pattern Editor** or click the **Launch Digital Pattern Editor** button on the TSM toolbar to open digital pattern project files in the Digital Pattern Editor. Additionally, you can use the Digital Pattern Project panel of the Test Program Editor to specify the pathname of the digital pattern project file to use in the test program.

The TSM Pin Map Editor and pin map schema natively support NI-Digital Pattern instruments.

Use the TSM Code Module API for the NI-Digital Pattern instruments to manage digital pattern instruments and sessions, to manage digital pattern waveform data, and to access digital pattern project files.

Note The NI-Digital Pattern VIs are available only in 64-bit LabVIEW.

You can create a basic test program from a digital pattern project. Select **Semiconductor Module»Create Test Program from Digital Pattern Project** to launch the Create Test Program from Digital Pattern Project dialog box.

Testing Multiple Sites in Parallel (TSM)

A semiconductor test program might need to test multiple DUTs at the same time in parallel to improve tester efficiency. A semiconductor test program can test one DUT at a time on a single test site or test multiple DUTs at a time on multiple test sites.

To implement multisite testing in a test program, the test engineer must consider the following requirements:

- The test program must be able to evaluate limits on multiple DUTs.
- The test program might execute individual test sites differently depending on previous test results specific to the individual test sites. For example, different sites might become disabled before a test begins depending on whether a handler can actually place DUTs in those sites, or different sites might become disabled during a test.
- The test program must be able to test DUTs on multiple sites simultaneously, even when multiple sites must simultaneously communicate with the same instrument.

TSM Implementation

The TestStand Batch and Parallel process models support multisite testing by creating a test socket for running a copy of the TestStand sequence in a new execution thread. However, the default TestStand behavior does not account for difficulties that a test engineer might encounter when programming for hardware shared among multiple test sites, such as NI-HSDIO instruments.

Use the TSM Code Module API to translate DUT pin names to instrument channels and sessions for the active sites and to publish test results to the active sites. The Semiconductor Multi Test step or the Semiconductor Action step creates the object reference `SemiconductorModuleContext` that you pass to a LabVIEW or .NET code module to use the TSM Code Module API. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system.

When you execute tests using the Batch process model, use the **Multisite Option** control on the Options tab of the Semiconductor Multi Test or the Semiconductor Action step to configure the following multisite execution options for the test:

- **One thread per subsystem**—Execute tests for each subsystem in a separate thread. A subsystem is a set of sites and system resources on the tester that operate independently of other sites and resources, typically because the sites share the same instrument, which requires the test program to test the sites together in a single thread. The Semiconductor Multi Test step or the Semiconductor Action step identifies subsystems by using the pin map

and the pins and relays shown in the **SemiconductorModuleContext Pins and Relays** control.

- **One thread only**—Execute tests for all sites in a single thread.
- **One thread per site**—Execute tests for each site in a separate thread. Use this option only when the code module does not use hardware shared among multiple sites.

The multisite option you select determines how many copies of a code module to execute. The more code modules that execute, the fewer sites TSM tests in any one code module.

Multisite Programming Techniques

When you create test programs to run on multiple sites, you must account for certain subsystem considerations, such as instrumentation resources, the relationship between the subsystem and the pin map, and using switches to share a channel between the same DUT pin on multiple sites. Consider the following issues during code module development:

- Pin and session queries
- Parallel For Loops
- Input parameters
- Specifications values
- Sharing data between code modules
- Publishing results
- Sharing instrument sessions between LabVIEW and .NET code modules

Note You cannot use multiple Semiconductor Multi Test steps or Semiconductor Action steps configured to use multiple threads in While loops, in Do While loops or in For loops that use the Custom Loop option when performing multisite testing. The steps report a run-time error in these situations. Use other types of loops instead, such as For loops that use the Fixed Number of Iterations option.

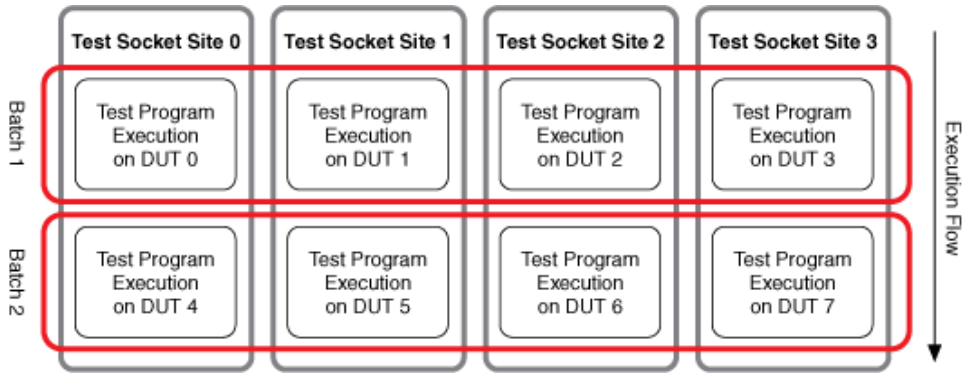
See Also

Multisite Scenarios Example

TSM Example Programs

Subsystem Considerations (TSM)

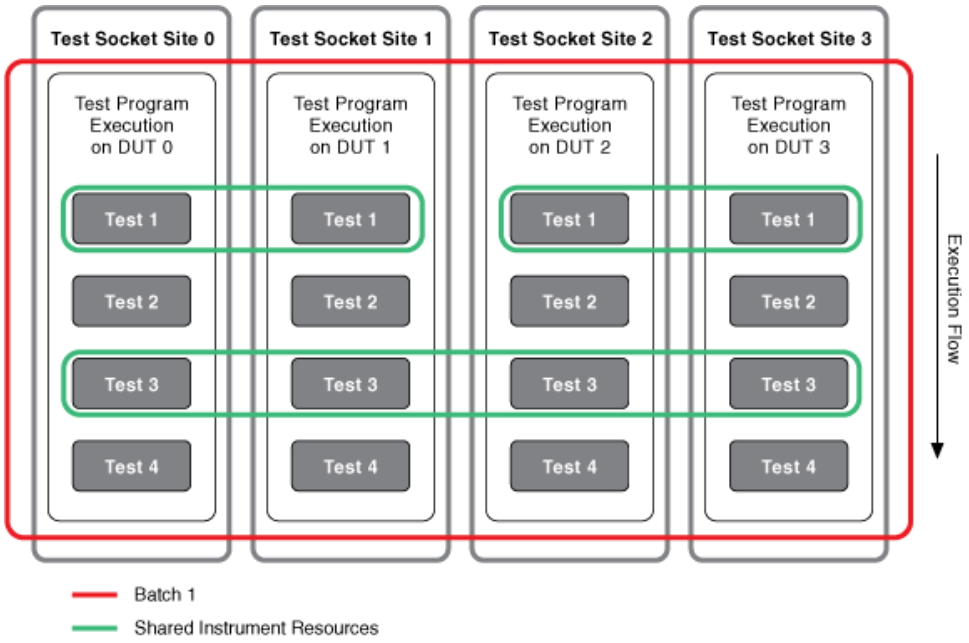
By default, the TestStand Batch process model executes a copy of the test program for each site simultaneously without changing any code by creating a test socket for each site. Each test socket runs in parallel and synchronizes at the beginning and end of the batch. However, each copy of the test program executes independently on each test socket and does not synchronize over each step within the test program execution shown in the following figure.



TSM site numbers do not always directly correspond to test socket indexes. Use the Get Site Runtime Data VI or the GetSiteRuntimeData .NET method of the TSM Application API or use the Get Test Information step to obtain specific site numbers.

Sharing Instrumentation Resources

Executing a copy of the test program for each socket works well when you use dedicated hardware for each DUT. In reality, however, it is more likely that multiple DUTs share the same instrumentation, which requires you to write test code to account for instrumentation sharing. In the following example, Test 1 and Test 3 share instrumentation resources.



Test 1 uses instrumentation shared between Sites 0 and 1 and between Sites 2 and 3. Test 3 uses instrumentation shared among all four sites. In this example, Sites 0 and 1 must synchronize at the first step, and Sites 2 and 3 must also synchronize at the first step. Because Sites 0 and 1 do not have to wait on Sites 2 and 3, each site group can continue testing after the first test is done. However, all sites must synchronize for Test 3 because of the dependency on instrumentation resources. After Test 3 completes, all sites can execute in parallel. At the end of the test program, all sites synchronize at the end of the batch to communicate with the handler or prober.

Using instrumentation multiplexed across sites or instruments that access multiple sites simultaneously introduce the following synchronization challenges

- **Multiplexed sharing**—You can use a multiplexer to connect a single instrument to multiple pins on the same DUT or on multiple DUTs. In this case, each site or pin must wait until an instrument is free.
- **Simultaneously sharing instruments**—You can connect a single instrument that supports simultaneous access to its resources to multiple pins on the same DUT or multiple DUTs, but for optimal performance you must synchronize the operations to occur at the same time. In this case, one code module can take measurements for multiple sites and report the

measurements back to each test socket. Code modules must account for taking measurements for multiple sites.

- **Relay sharing**—Run any sites that share relays in the same subsystem.

See Also

Grouping Instruments

Subsystems and Pin Maps (TSM)

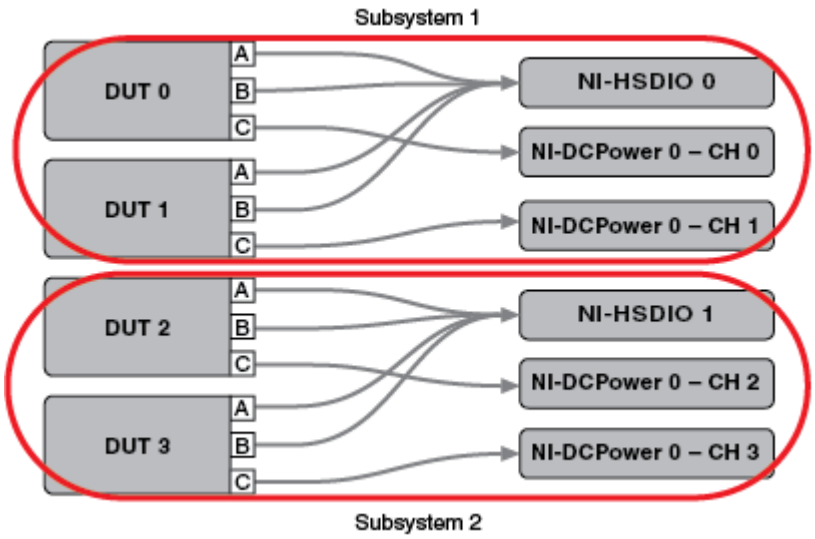
The Semiconductor Multi Test and the Semiconductor Action steps examine the pin map to determine which sites to synchronize. Steps can also specify required pins or pin groups and relays, relay groups, or relay configurations for a code module. For example, if you connect a pin to a shared instrument but a code module does not require the pin, you might be able to execute the code in parallel. After determining the sites to synchronize, the Semiconductor Multi Test and the Semiconductor Action steps wait until those sites reach the synchronization point or become disabled. The step then calls a LabVIEW or .NET code module and passes a `SemiconductorModuleContext` object that contains the information about the sites for the code module.

A **subsystem** is a part of the test system that can operate independently. TSM defines subsystems dynamically based on the active sites, the flow of each test socket, and the instrumentation required to conduct a test.

The following figure illustrates the connections between four DUTs/sites, two NI-HSDIO instruments, and one power supply. Each NI-HSDIO instrument requires that the channels for that instrument operate together but the power supply can operate four independent channels. Each DUT has an A, B, and C pin. Each pin A and pin B connects to an NI-HSDIO instrument, and each pin C connects to the power supply.

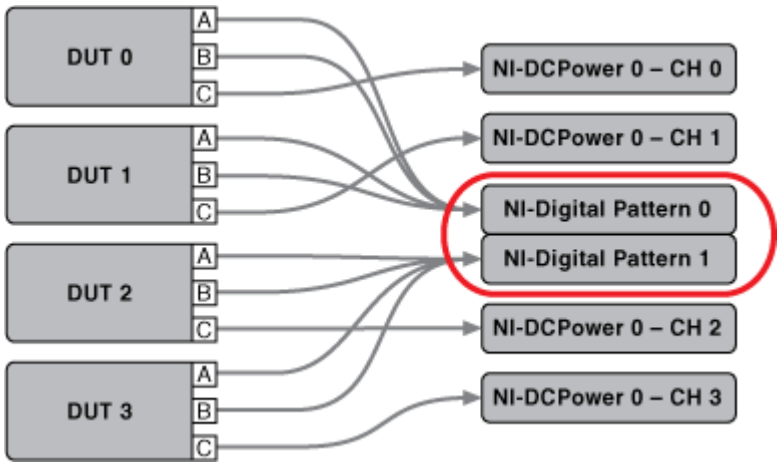
The two subsystems illustrated in the figure can operate independently. If a test uses pin A or B, you can break the test up into two subsystems. If a test uses only pin C, you can break the test setup into four subsystems—one for each site.

By default, the Semiconductor Multi Test and Semiconductor Action steps assume that a code module uses every DUT pin and no site relays. Improve the performance of the tester by specifying only the pins that the code module uses.



The following figure illustrates a similar situation as the previous figure but replaces the two NI-HSDIO instruments with two NI-Digital Pattern instruments that are grouped together. By default, TSM groups all NI-Digital Pattern instruments together.

The figure shows a single subsystem for test steps that use only pins A or B. TSM executes the test code module in a single TestStand test socket because all NI-Digital Pattern channels must operate together. The NI-Digital Pattern driver performs some operations on the channels in parallel to achieve improved performance. As in the previous example, if a test uses only pin C, you can break the test setup into four subsystems.



Multisite Programming with Switches (TSM)

When you must share a channel between the same DUT pin on multiple sites, use a switch to control which pin the device is connected to when executing a test. The Switching example demonstrates how to write a test program that uses multiplexed routes in a multisite test program. The example uses an Switch Executive Virtual Device, but you can use the same programming methods with any switch instrument.

Data Management Recommendations (TSM)

The following table summarizes available data mechanisms in TestStand, in the TestStand Semiconductor Module (TSM), and in the programming environment you use. Refer to the following Data Mechanisms Selection section for more information about issues to resolve before selecting a data mechanism.

Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

Type of Data	Used Only in TestStand	Used Only in Code Modules	Used in TestStand and in Code Modules
Per-site	Sequence local variables and parameters ▼ Details TSM executes a unique run-time sequence for each site. Use the TestStand sequence local variables and parameters to create per-site variables in TestStand.	Site Data API (TSM Code Module API) ▼ Details Because TSM can execute a code module on multiple sites in parallel, managing per-site data in a code module can be complex. Use the use the Set Site Data VI or `SetSiteData` .NET method and the Get Site Data VI or `GetSiteData` .NET method in the TSM Code Module API to get and set arbitrary per-site data in a code	▪ TestStand—Semiconductor Action step parameter ▪ Code Module—TSM Site Data API ▼ Details Because TestStand and TSM code modules use different data mechanisms for storing per-site data, you must pass the data between TestStan

			module. With the site data API, you associate each per-site data value with a data ID at run time to uniquely identify the data with a name.	d and each code module to share data. To pass per-site data stored in a TestStand variable into a code module, use the Set Site Data VI or `SetSiteData` .NET method and the Get Site Data VI or `GetSiteData` .NET method in the TSM Code Module API to create a separate code module to store the data and call the code module from a Semiconductor Action step configured to use the **One thread per site** value in the **Multisite Option** control. Pass the unique data ID and the TestStand variable as an array of one element into the code module to store the variable value in per-site data for code modules to use. To pass per-site data stored in a code module into TestStand, use the Set Site Data VI or `SetSiteData` .NET method and the Get Site Data VI or `GetSiteData` .NET method in the TSM Code Module API to create a separate code module to get the data and call the code module from a Semiconductor Action step configured

			to use the **One thread per site** value in the **Multisite Option** control. The code module must extract the data from the first element of the array the set data API in the TSM Code Module API returns. Pass the unique data ID and the TestStand variable to receive the data in the code module.
Global	Sequence file global variable (after changing **Sequence File Globals** option to **All Executions Share the Same File Globals**) ▾ Details You can use sequence file global variables to manage global data in TestStand, but you must change the default value of the **Sequence File Globals** option on the General tab of the Sequence File Properties dialog box from **Separate File Globals for Each Execution** to **All Executions Share the Same File Globals** to ensure that the sequence file global variables are shared for all sites. If you do not change the value of this option in the Sequen	Language specific data, Global Data API (TSM Code Module API) ▾ Details For global data used only in code modules, use the data mechanism that the language the code module uses to achieve the best performance. For example, use LabVIEW global variables or functional global variables to share global data among LabVIEW code modules. You can also use the Set Global Data VI or `SetGlobalData` .NET method and the Get Global Data VI or `GetGlobalData` .NET method in the TSM Code Module API.	▪ TestStand—Sequence file global variable (after changing **Sequence File Globals** option to **All Executions Share the Same File Globals**) ▪ Code Module—Step module parameter ▾ Details To use global data in code modules and in TestStand, use sequence file global variables to manage the data in TestStand. You must change the default value of the **Sequence File Globals** option on the General tab of the Sequence File Properties dialog box fr

	ce File Properties dialo g box, the sequence file global variables store p er-site data only.		om **Separate File Glo bals for Each Executi on** to **All Executions Share the Same File Globals** to ensure that the sequence file globa l variables are shared fo r all sites. If you do not change the value of this option in the Sequence File Properties dialog b ox, the sequence file gl obal variables store per -site data only. Within the code modul es, use the data mecha nism that the language the code module uses t o achieve the best perf ormance. For example, use LabVIEW global vari ables or functional glob al variables to share glo bal data among LabVIE W code modules. You c an also use the Set Glo bal Data VI or `SetGlo balData` .NET metho d and the Get Global Da ta VI or `GetGlobalDa ta` .NET method in the TSM Code Module API. Pass the global data int o and out of code mod ules using parameters t o the code modules.
Global constants	Test conditions ▾ Details You can use the same mechanism described i	Specifications ▾ Details You can use the same d ata mechanism describ	▪ TestStand—Tes t conditions

	n the Global data – Used Only in TestStand table cell. Alternatively, you can define test conditions in the Test Program Editor. You can use the Get Test Information step to reference the test conditions in a sequence. You can create multiple test program configurations that use different values for each test condition that you define. Test conditions support only numbers, strings, and Boolean values.	ed in the Global data – Used Only in Code Modules table cell. Alternatively, you can use a specifications file to define named expressions that evaluate to constants at run time. Use the Digital Pattern Editor to edit specifications files. Use the TSM Code Module API to get the value of a specification. Specifications support only numbers.	▪ Code Module—Step module parameter ▾Details To use global constants in code modules and in TestStand, use test conditions or a sequence file global variable to define and access the constant in TestStand. Pass the global constant into a code module using parameters to the code modules.

TSM site numbers do not always directly correspond to test socket indexes. Use the Get Site Runtime Data VI or the GetSiteRuntimeData .NET method of the TSM Application API or use the Get Test Information step to obtain specific site numbers.

Data Mechanisms Selection

Before selecting a data mechanism to use, you must understand how and where the test program uses the data. Consider the following questions as you select a data mechanism:

- How should the test program access the data? Some data mechanisms work best in code modules, and others work best in TestStand. Sharing data between code modules and TestStand is more complicated than using the data in only one environment or the other. Refer to the Accessible Locations column in the following table.
- Where should the test program define the data? Some data mechanisms define the data in an external file that you access from TestStand or from code modules. Some data mechanisms define the data while editing the test program, and others create the data at run time. Refer to the Defined Location column in the following table.

- What type of data does the test program use? Some data mechanisms are limited to specific types of data, such as numbers, strings, and Boolean values. Refer to the Data Types column in the following table.
- Does the data remain constant throughout testing or does the test program modify the data? Refer to the Constant or Dynamic column in the following table.
- Does each site use unique data values or do all sites share the same data values? Refer to the Per-Site or Global column in the following table.

Data Mechanisms Table

The following table outlines the different mechanisms for managing data in TestStand, in TSM, and in the programming environment you use.

Data Mechanism	Purpose	Accessible Locations	Defined Location	Data Types	Constant or Dynamic	Per-Site or Global	Limitations
Language specific data (LabVIEW global variables or .NET variables)	Manage per-site or global data in code modules	Code modules	Code module	Any language-specific type	Dynamic	Per-site or global	Difficult to manage per-site data efficiently
Site Data API (TSM Code Module API)	Manage per-site data in code modules	Code modules	Code module	Any language-specific type, except LabVIEW classes	Dynamic	Per-site	Does not support LabVIEW classes
Global Data API (TSM Code Module API)	Manage global data in code modules	Code modules	Code module	Any language-specific type, except LabVIEW classes	Dynamic	Global	Does not support LabVIEW classes; potentially slower than language-specific data
Step Input Data API (T	Pass per-site data from TestStan	TSM steps, code modules	TestStand variable	Numbers, strings, and	Dynamic	Per-site	Supports only numbers, strings,

SM Code Module API)	d into a code module			Boolean values			and Boolean values
Step module parameters	Pass global data from TestStand into a code module	TSM steps, code modules	TestStand variable	Any TestStand type	Dynamic	Global	—
Sequence local variables and parameters	Manage per-site data in TestStand	TestStand	TestStand variable	Any TestStand type	Dynamic	Per-site	—
Sequence file global variables	Manage per-site data or global data in TestStand	TestStand	TestStand variable	Any TestStand type	Dynamic	Per-site or global, depending on a sequence file setting	Supports per-site or global data but not both; requires changing a sequence file setting to support global data instead of per-site data
Specifications	Specify constant expressions that code modules and digital timing and levels sheets use	Digital Pattern Editor, code modules	Specifications file	Numbers	Constant	Global	Supports only numbers
Test Conditions	Specify constant test conditions associated with current test configuration	Test Program Editor, TSM Operator Interface	Test Program Editor	Numbers, strings, and Boolean values	Constant	Global	Supports only numbers, strings, and Boolean values

Station and Lot Settings	Specifies settings associated with station or current lot	TSM Operator Interface	TestStand type	Any TestStand type	Constant	Global	—

Code Module Development (TSM)

The Semiconductor Multi Test and the Semiconductor Action steps call a single code module for all the sites that synchronize at a certain location, which means the code module executes on multiple sites at the same time. The Semiconductor Multi Test or Semiconductor Action step creates a `SemiconductorModuleContext` object that contains the information about the sites on which the code module executes. Pass the `Step.SemiconductorModuleContext` property to the code module to use the TSM Code Module API.

Code Module Development Guidance

As you develop each part of the test program, consider the functionality that belongs in code modules and the functionality that belongs in test sequences. In general, use the following guidelines:

- **Code Modules**—Use code modules written in LabVIEW or in .NET for instrument driver calls to configure instruments and acquire data. Additionally, use code modules for other code necessary to perform a specific DUT test or other specific actions so you can reuse the code modules in other test programs.
- **Test Sequences**—Use test sequences to call code modules and evaluate results the code modules generate.

Use Test Sequence for Test Flow

Use the TSM test sequence instead of code modules to determine test flow. For example, if the test program requires you to execute a certain test only if a previous test fails, ensure that the test sequence contains the logic for executing tests in a specific order. Use this development strategy to make the flow of the test program

more apparent to other test engineers and to help generalize code modules for potential reuse in other test programs.

Evaluate Results in TSM

Use code modules to execute specific tests, but do not use code modules to evaluate numeric results of tests. Use the Publish Data TSM Code Module API to transfer test measurement data to the Semiconductor Multi Test step. Configure the Semiconductor Multi Test step to evaluate the measurement against a set of limits or a specific value. Because test limits might change throughout the life cycle of a DUT, evaluating the measurements in TSM rather than in code modules enables you to more easily adjust the test limits as needed. Additionally, using this technique helps to generalize code modules for potential reuse in other test programs.

You might need to perform calculations on the test measurements before evaluating the data because TSM does not offer the functionality you need or because you need to attain maximum performance for a processor-intensive calculation. For example, use a code module to calculate the FFT of a waveform, and use a Semiconductor Multi Test step in TSM to perform the final test evaluation.

Implement Loops in Code Modules for High Performance

Implement loops in code modules when execution speed is critical. For example, if a test needs to sweep an instrument parameter over a range of values as quickly as possible, use a For loop in a code module rather than executing the loop in TSM. Additionally, using parallel For Loops in LabVIEW code modules to apply the same settings to multiple instruments at once is much faster than implementing loops in TSM.

Use Per-Site Data to Store Data within Code Modules

Use the Set Site Data VI or `SetSiteData` .NET method and the Get Site Data VI or `GetSiteData` .NET method in the TSM Code Module API to store data that you need to access in another code module later. Using this development strategy provides an advantage over storing data in TSM variables because the API maintains unique data for each site. Additionally, you can access the data directly within code modules instead of passing the data from TSM as a parameter.

Specify Pin Names as Parameters to Code Modules

Use parameters to pass pin names into code modules from TSM instead of hard coding pin names within the code modules. Using this technique improves the readability of the test program by making the use of each pin in the code module more apparent to test engineers. It also generalizes code modules for potential reuse in other test programs.

Other Suggestions

Additionally, consider the following issues during code module development:

- Querying pins and sessions
- Using parallel For Loops
- Grouping Instruments
- Specifying input parameters
- Obtaining values from specifications files
- Sharing data between code modules
- Publishing results
- Sharing instrument sessions between LabVIEW and .NET code modules
- Using LabVIEW VI Analyzer
- Using LabVIEW Classes

See Also

TSM Code Module API

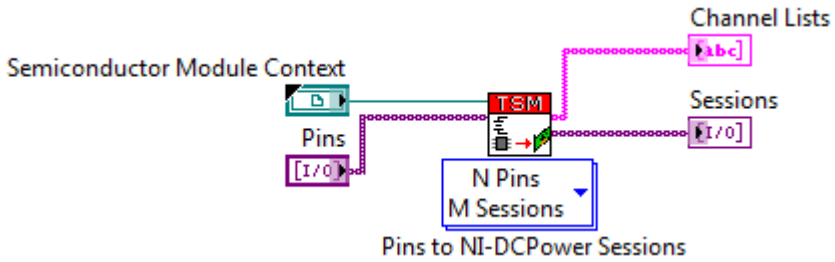
Pin and Session Queries (TSM)

NI instrument drivers refer to sessions and channel lists rather than to sites and pins. When you take a measurement using an NI instrument driver, you must first perform a pin query to look up the associated instrument sessions and channel lists.

To conduct a pin query, you must first know the type of instrument to which the pins are connected. Each instrument type has a corresponding pin query VI or .NET method.

The following figure and code snippet shows how to use the Pin(s) To NI-DCPower Session(s) polymorphic VI or the `GetNIDCPowerSessions` .NET method to obtain the associated sessions and channel lists for a set of pins using an NI-DCPower instrument.

LabVIEW



.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string dcPowerPins)
{

NIDCPower[] dcPowerSessions;
string[] dcPowerChannelStrings;
semiconductorModuleContext.GetNIDCPowerSessions(dcPowerPins, out
dcPowerSessions, out dcPowerChannelStrings);

}
```

The Semiconductor Module context provides information to the VI about which sites the subsystem includes. The pin query returns an array of instrument sessions and channel lists. Each element with the same index in each array is associated. The pin query returns the order of the sessions and channels in a deterministic manner based on the active sites and pins.

Different instrument types might return different outputs depending on the information required to control the instrument. For example, some instruments do not have channels and thus do not return a channel list. Other instruments might return multiple session arrays.

LabVIEW Polymorphic Instances of Pin(s) to Session(s) VIs

The Pin(s) to Session(s) VIs offer a subset of the following polymorphic instances:

- Multiple Pins - Multiple Instruments or Sessions
- Single Pin - Multiple Instruments or Sessions
- Multiple Pins - Single Instrument or Session
- Single Pin - Single Instrument or Session

Depending on the instrument type, some of these options might not be available. If you operate on multiple pins, you must select the Multiple Pins options. If you operate on a single pin, select the Single Pin options. You can build the pin name into an array of strings and use the Multiple Pin options if a Single Pin option is not available for an instrument type.

You can use different instances of the same VI in multiple locations of a code module. For example, if you want to set the voltage of all pins to `0`, you can use the Multiple Pins - Multiple Sessions instance to access the instrumentation. If you then want to force a current on a single pin, use the Single Pin - Multiple Sessions instance of the VI.

Note NI does not recommend deconstructing the outputs of the VI to obtain information about each site or pin. Instead, use the Get Session and Channel Index VI to obtain the sessions and channel lists.

.NET Method Overloads

The .NET API provides overloads for the pin query methods. The overloaded parameters allow you to specify a single pin or an array of pins and output a single instrument session or an array of instrument sessions. The type of the return value depends on which overload you use and provides a publish method that matches the pin query method overload. Depending on the instrument type, some method overloads might not be available.

You can use different method overloads in multiple locations of a code module. For example, if you want to set the voltage of all pins to 0, you can use the overload that takes an array of pins and outputs an array of sessions to access the

instrumentation. If you then want to force a current on a single pin, use the overload that takes a single pin and outputs an array of sessions.

Note NI does not recommend deconstructing the outputs of the method to obtain information about each site or pin. Instead, use the `GetSessionAndChannelIndex` .NET method on the pin query context object returned from the pin query method to obtain the sessions and channel lists.

Single Sessions or Multiple Sessions

Select a Single Session VI instance or the .NET method overload that outputs a single session only under the following conditions:

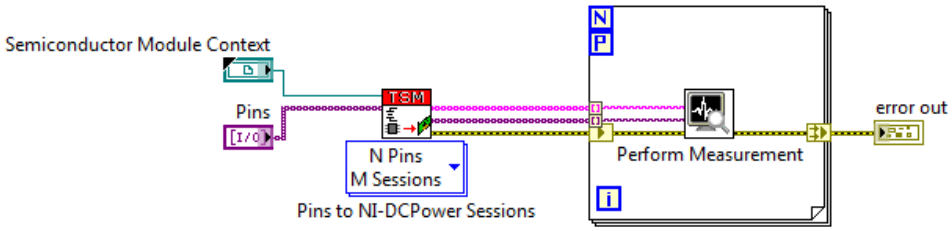
- You are using a NI-Digital Pattern pin query and all NI-Digital Pattern instruments belong to the same group in the pin map file. By default, TSM groups all NI-Digital Pattern instruments together.
- You are using a NI-SCOPE pin query and all NI-SCOPE instruments belong to the same group in the pin map file. By default, TSM groups all NI-SCOPE instruments together.
- You are using a NI-DCPower pin query and all NI-DCPower channels belong to the same group in the pin map file. By default, TSM groups all NI-DCPower channels together.
- You use only a single session of a given type for the tester.
- You know that the instruments of that type are and will always be the limiting factor when creating a subsystem.

Select a Multiple Sessions VI instance or use the .NET method overload that outputs an array of sessions if you are not sure whether to return multiple sessions or a single session. Using this technique helps you avoid having to modify code modules if a pin map changes or if you convert a test program to run on other testers or with other pin maps. Using a Single Session VI instance or the .NET method overload that outputs a single session with a set of pins and sites that returns multiple sessions results in a run-time error.

Parallel For Loops (TSM)

When you obtain multiple sessions, you will usually use a parallel For Loop to control the instruments in parallel to increase multisite efficiency. You do not need to use a parallel For Loop for NI-Digital Pattern, NI-SCOPE, or NI-DCPower pin queries if all instruments or channels belong to the same group in the pin map file. By default, TSM groups all NI-Digital Pattern instruments into the same NI-Digital Pattern instrument group, all NI-SCOPE instruments into the same NI-SCOPE instrument group, and all NI-DCPower channels into the same NI-DCPower channel group. You can edit instrument or channel groups in the Pin Map Editor.

LabVIEW



Right-click a For Loop and select **Configure Iteration Parallelism** from the context menu to create a parallel For Loop. Index the session and channel list arrays in the For Loop to access each instrument in parallel.

Error Handling with Parallel For Loops

If you enable parallelism in a For Loop, any shift registers on error wires automatically become error registers, which allow for errors to be merged across all iterations of the For Loop, as illustrated in the previous graphic.

Because you cannot use shift registers in parallel For Loops, you must build error clusters into an array and then merge the errors. Complete the following steps to structure error wires so that you can handle all potential errors.

1. If no iterations of the parallel For Loop execute, which can occur when a previous error exists, the loop does not record any errors you pass into the loop. To avoid this situation, branch the error wire before it enters the parallel For Loop and connect the new segment around the loop to the top input terminal of the Merge Errors node. Using this wire connection strategy ensures

that you can track errors that occurred earlier during execution outside of the loop.

2. Enable indexing for the error output tunnel exiting the loop. If you do not enable indexing, only the error information from the final iteration of the loop is maintained.
3. Use the Merge Errors node to capture error information from the code that executed before the loop execution and also each of the loop iterations.

.NET (C#)

One option for parallel For Loops in .NET is the System.Threading.Tasks.Parallel.For method. The following example demonstrates how to use this method to perform a measurement with each instrument session in parallel.

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string dcPowerPins)
{

NIDCPower[] dcPowerSessions;
string[] dcPowerChannelStrings;
semiconductorModuleContext.GetNIDCPowerSessions(dcPowerPins, out
dcPowerSessions, out dcPowerChannelStrings);
var measurements = new double[dcPowerSessions.Length];
Parallel.For(0, dcPowerSessions.Length, i =>
{

measurements[i] = PerformMeasurement(dcPowerSessions[i],
dcPowerChannelStrings[i]);

});

}
```

See Also

Enable Parallel For Loop Iterations in VIs

Grouping Instruments or Channels (TSM)

You can group certain instruments or channels together and treat them as a single entity. When all the instruments of the same type belong to a single group or when the instruments of the same type in a subsystem belong to a single group, you can use the versions of the pin query VIs and .NET methods that return a single session and you do not need to use parallel For Loops to iterate over the instrument driver sessions. The instrument driver specific to the grouped instruments performs most operations on all channels in parallel to achieve improved multisite efficiency. Refer to the instrument driver help for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

Grouping Instruments with NI-Digital Pattern Driver

By default, when you create a new NI-Digital Pattern instrument in the pin map file, the NI TestStand 2019 Semiconductor Module (TSM) and later set the `group` attribute to `Digital` so that all newly created NI-Digital Pattern instruments belong to the same group. TSM creates a single session for each group of NI-Digital Pattern instruments in the pin map file. TSM 2017 and earlier do not automatically group NI-Digital Pattern instruments together in pin map files. Use the Pin Map Editor to modify existing pin map files to change the value of the **Group** option for each instrument to assign the instrument to the same group.

Grouping Instruments with NI-SCOPE Driver

By default, when you create a new NI-SCOPE instrument in the pin map file, the TSM 2019 and later set the `group` attribute to `Scope` so that all newly created NI-SCOPE instruments belong to the same group. TSM creates a single session for each group of NI-SCOPE instruments in the pin map file. TSM 2017 and earlier do not automatically group NI-SCOPE instruments together in pin map files. Use the Pin Map Editor to modify existing pin map files to change the value of the **Group** option for each instrument to assign the instrument to the same group.

Grouping Channels with the NI-DCPower Driver

By default, when you create a new NI-DCPower instrument in the pin map file, TSM 2020 and later will create a single channel group containing all instrument channels.

TSM creates a single session for each channel group of NI-DCPower instruments in the pin map file. TSM 2019 and earlier do not allow for channel grouping and pin maps created with older versions of TSM do not contain channel group information. Complete the following steps to convert all NI-DCPower instruments in the pin map to use channel groups:

1. Open the Pin Map Editor.
2. Select the NI-DCPower instrument in the **Instruments** section on the Pin Map tab.
3. Click the **Convert NI-DCPower Instruments** button.
4. Click **Yes** in the Convert all NI-DCPower Instruments dialog box to complete the conversion.

Notes

- To use NI-DCPower channel groups you will need the correct combination of TSM 2020 and NI-DCPower Driver 2020. Both the driver and TSM are fully backwards compatible but the methods used and VI calls made by TSM and the driver must be changed when channel groups are used.
- TSM does not allow a combination of NI-DCPower instruments with channel groups and NI-DCPower instruments without channel groups.

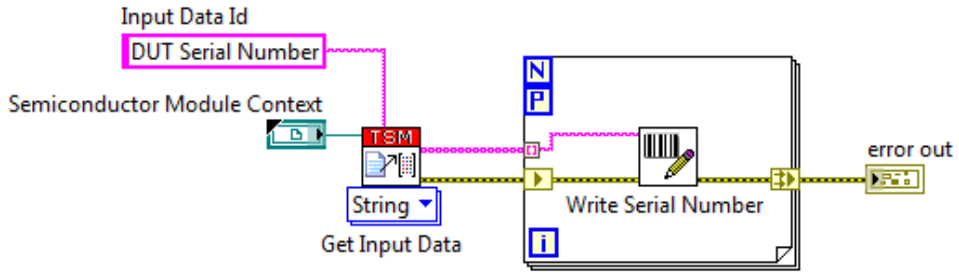
Input Parameters (TSM)

Use the Parameter Table of the Module tab to pass the same parameter values for all sites from the sequence file directly into a code module.

In some cases, you might want to use different values for each site, such as setting different register values when trimming or using different calibration values for each site. Use the Per-Site Inputs tab of the Semiconductor Multi Test or the Semiconductor Action step to specify different parameter values within each test sequence for each site. Use the Get Input Data VI or the `GetInputDataAsBooleans`, `GetInputDataAsDoubles`, or `GetInputDataAsStrings` .NET methods

in a code module to obtain the appropriate values for each site. These VIs and .NET methods return an array of values—one for each site that is part of the subsystem calling the code module.

LabVIEW



.NET (C#)

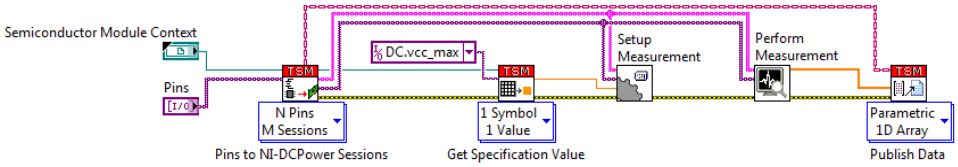
```
public static void ExampleCodeModule (ISemiconductorModuleContext
semiconductorModuleContext)
{
var serialNumbers =
semiconductorModuleContext.GetInputDataAsStrings(inputDataId: "DUT Serial
Number");
Parallel.For(0, waveformInputData.Length, i =>
{
WriteSerialNumber(serialNumbers[i]);
});
}
```

Getting Values for Specifications (TSM)

Specification files (`.specs`) define a set of variables and associated numeric values that you reference in code modules to configure hardware or provide input to tests procedures. Use the Get Specification(s) Value(s) VI or the `GetSpecificationsValue` or `GetSpecificationsValues` .NET methods to query one variable or an array of variables and return the calculated value or values. The VI and methods

return an error or exception when you reference a variable that does not exist in the specifications file, which can result in a run-time error when the sequence executes.

LabVIEW



.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string[] pins)
{

NIDCPower[] dcPowerSessions;
string[] channelStrings;
var pinQuery = semiconductorModuleContext.GetNIDCPowerSessions(pins, out
dcPowerSessions, out channelStrings);
double vccMax =
semiconductorModuleContext.GetSpecificationsValue("DC.vcc_max");
SetupMeasurement(dcPowerSessions, channelStrings, vccMax);
var results = PerformMeasurement(dcPowerSessions, channelStrings);
pinQuery.Publish(results);

}
```

Organization of Measurement Data (TSM)

Sometimes test code needs to process, make calculations with, or otherwise manipulate measurement data on a per-site basis before publishing it to TestStand tests for evaluation. However, typical TSM test code modules produce data organized in an array indexed by instrument session instead of by site. Use the following TSM VIs and .NET methods to convert per-instrument data to per-site data. TSM always orders per-site data in the same order as the sites returned by the Get Site Numbers VI or SiteNumbers .NET property.

- **Extract Pin Data VI and ExtractPinData .NET method**—Extract measurement data for a single pin in the pin query from per-instrument measurement data and return per-site data. If you are using pin groups in a pin query and want to extract pin data for each pin in the group, you can use the Get Pins in Pin Group VI or `GetPinsInPinGroup` .NET method to convert a pin group into an array of pin names.
- **Per-Instrument to Per-Site Data VI and PerInstrumentToPerSiteData .NET method**—Convert per-instrument measurement data to per-site data in an array organized by site and pin.
- **Per-Instrument to Per-Site Pattern Results VI and PerInstrumentToPerSitePatternResults .NET method**—Convert per-instrument digital pattern results obtained from the NI-Digital Pattern Driver into per-site data.
- **Per-Instrument to Per-Site Waveforms VI and PerInstrumentToPerSiteWaveforms .NET method**—Convert per-instrument digital pattern waveforms captured by the NI-Digital Pattern Driver into waveforms organized by site.
- **Per-Site to Per-Instrument Waveforms VI and PerSiteToPerInstrumentWaveforms .NET method**—Convert per-site waveforms into waveforms organized by instrument that you can use as source waveforms with the NI-Digital Pattern Driver.

To publish per-site data to TestStand tests for evaluation, use the site-based polymorphic instances of the Publish Data VI and the `PublishPerSite` .NET method on the `ISemiconductorModuleContext` interface. These VIs and .NET methods expect per-site data as inputs. Like the other Publish Data VIs and `Publish` .NET methods, these VIs and .NET methods attach the per-site data to the appropriate test on the TestStand steps for evaluation.

See Also

Publishing Results

Sharing Data between Code Modules

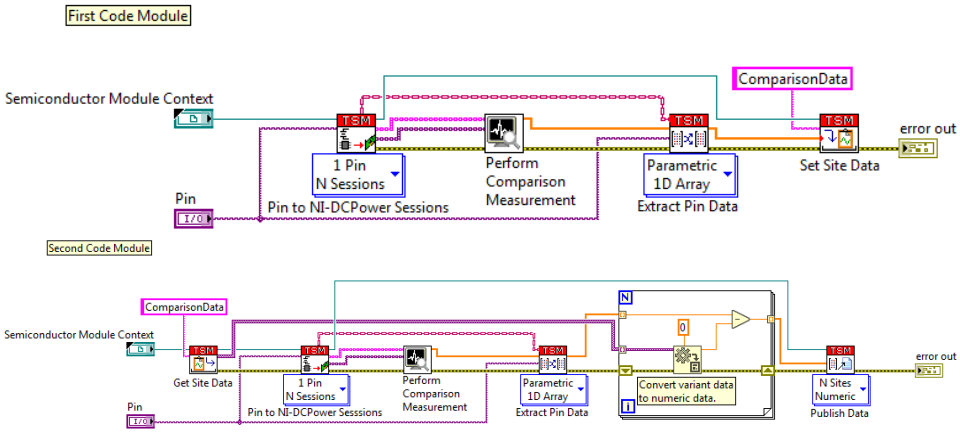
Sharing Data between Code Modules (TSM)

Data can be stored in the SemiconductorModuleContext in one code module and retrieved later in another code module using an ID string. The data can be stored as a single data value available globally to all sites or as a data value per site. Use the Set Global Data VI or `SetGlobalData` .NET method and the Get Global Data VI or `GetGlobalData` .NET method to store and retrieve data shared by all sites. Or use the Set Site Data VI or `SetSiteData` .NET method and the Get Site Data VI or `GetSiteData` .NET method to store and retrieve site specific data.

Note When setting site specific data with the Set Site Data VI or the `SetSiteData` .NET method, the data should be ordered to match the order of sites in the Semiconductor Module context. This order might not be sequential. If the data you are storing was acquired from instruments using a pin query, use the Extract Pin Data VI or the `ExtractPinData` method on the `PinQueryContext` .NET object to extract the data from the measurements array into the correct order for the Set Site Data VI or `SetSiteData` .NET method. Otherwise, use the Get Site Numbers VI or the `SiteNumbers` property on the SemiconductorModuleContext .NET object to determine the order of the sites in the Semiconductor Module context and arrange the data manually.

The following example shows how to store a per-site measurement data for comparison in a later test step:

LabVIEW



.NET (C#)

```
public static void FirstCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string pin)
{

NIDCPower[] dcPowerSessions;
string[] channelStrings;
var pinQuery = semiconductorModuleContext.GetNIDCPowerSessions(pin, out
dcPowerSessions, out channelStrings);
var results = PerformComparisonMeasurement(dcPowerSessions, channelStrings);
var perSiteData = pinQuery.ExtractPinData(results);
semiconductorModuleContext.SetSiteData("ComparisonData", perSiteData);

}
public static void SecondCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string pin)
{

var siteDataObjects =
semiconductorModuleContext.GetSiteData("ComparisonData");
var perSiteComparisonData = siteDataObjects.Cast<double>().ToArray();
NIDCPower[] dcPowerSessions;
string[] channelStrings;
var pinQuery = semiconductorModuleContext.GetNIDCPowerSessions(pin, out
dcPowerSessions, out channelStrings);
```

```
var results = PerformComparisonMeasurement(dcPowerSessions, channelStrings);
var perSiteData = pinQuery.ExtractPinData(results);
double[] comparisonResult = new double[perSiteData.Length];
for (int dataIndex = 0; dataIndex < perSiteData.Length; dataIndex++)
{
comparisonResult[dataIndex] = perSiteData[dataIndex] -
perSiteComparisonData[dataIndex];
}
semiconductorModuleContext.PublishPerSite(comparisonResult);
}
```

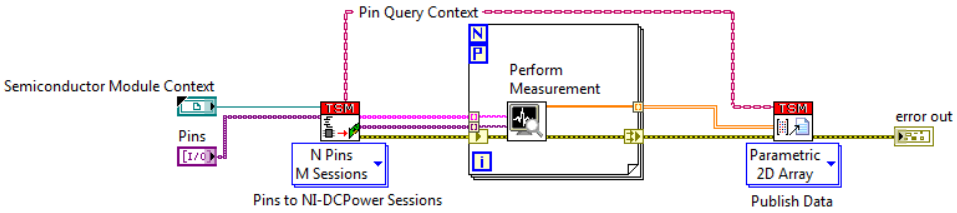
Publishing Results (TSM)

After you take measurements, you typically publish the results to the test sockets to evaluate and log. It might become difficult to know which piece of data to associate with each site and pin because you typically take measurement using multiple instruments connected to multiple DUTs and return arrays with the measurements mixed among each site. Use the appropriate Pin Query VI or .NET method with the Publish Data VI or `Publish` .NET method to return data to the correct site. Because the Publish Data VI and `Publish` .NET method manage arrays of different dimensions and extract the required data in the required order, you do not need to manage arrays before publishing measurements. The Pin Query VI or .NET method returns a Pin Query Context to associate a pin query operation with a matching instance of the publish operation.

Note The Publish Data VI and Publish .NET method have no effect in code modules you call from the Semiconductor Action step because no test locations exist for publishing data.

The following figure shows how to publish pin-based data from a measurement taken with an NI-DCPower instrument.

LabVIEW



.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string[] pins)
{

NIDCPower[] dcPowerSessions;
string[] dcPowerChannelStrings;
var pinQuery = semiconductorModuleContext.GetNIDCPowerSessions(pins, out
dcPowerSessions, out dcPowerChannelStrings);
var measurementResults = new double[dcPowerSessions.Length];
Parallel.For(0, dcPowerSessions.Length, i =>
{

measurementResults[i] = PerformMeasurement(dcPowerSessions[i],
dcPowerChannelStrings[i]);

});
pinQuery.Publish(measurementResults);

}
```

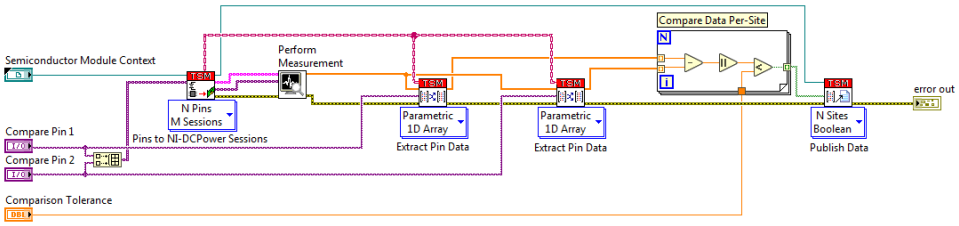
In C# use the `var` keyword to declare the variable to store the Pin Query Context return value of the Pin Query method with a dynamic type. Using this technique can make it easier to select the correct Pin Query Context type based on which overload of the Pin Query method you select.

Per-Site Publishing

You can also use a site-based instance of the Publish Data VI or the `Publish` method on the SemiconductorModuleContext .NET object to publish values for each site. Use the optional **Pin** and **Published Data Id** parameters of the Publish Data VI

or `PublishPerSite` .NET method to publish data for each site to tests that have non-empty **Pin** and **Published Data Id** fields.

LabVIEW



.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string comparisonPin1, string comparisonPin2,
double comparisonTolerance)
{

NIDCPower[] dcPowerSessions;
string[] channelStrings;
string[] pins = {comparisonPin1, comparisonPin2};
var pinQuery = semiconductorModuleContext.GetNIDCPowerSessions(pins, out
dcPowerSessions, out channelStrings);


var results = PerformComparisonMeasurement(dcPowerSessions, channelStrings);


var perSiteDataForPin1 = pinQuery.ExtractPinData(results, comparisonPin1);
var perSiteDataForPin2 = pinQuery.ExtractPinData(results, comparisonPin2);


int numSites = semiconductorModuleContext.SiteNumbers.Count;
bool[] comparisonResult = new bool[numSites];
for (int siteIndex = 0; siteIndex < numSites; siteIndex++)
{
```

```
comparisonResult[siteIndex] = Math.Abs(perSiteDataForPin1[siteIndex] -
perSiteDataForPin2[siteIndex]) < comparisonTolerance;
}
semiconductorModuleContext.PublishPerSite(comparisonResult);
}
```

Sharing Natively Supported Instrument Sessions between LabVIEW and .NET Code Modules (TSM)

Note This topic applies to instruments that TSM natively supports. Sharing custom instrument sessions between LabVIEW and .NET code modules requires a custom solution.

When you write a test program that uses the same instrument session in LabVIEW and in .NET code modules, use the TSM Code Module API methods for storing and accessing instrument sessions from the pin map associated with the test program because the TSM Code Module API methods properly store and retrieve the instrument sessions so you can access the sessions by LabVIEW and .NET code modules.

Execute Code Modules in the Same Process that Creates Instrument Sessions

Code modules that access instrument sessions must execute in the same process as the code module that created the session. .NET code modules execute in the process that executes the test program. LabVIEW code modules execute in the process that executes the test program or in the LabVIEW Development System process, depending on the LabVIEW Adapter settings. .NET code modules cannot use instrument sessions created by LabVIEW code modules running in the LabVIEW Development System process. LabVIEW code modules running in the LabVIEW Development System cannot use instrument sessions created by .NET code modules.

LabVIEW Adapter Configuration

When you share instrument sessions between LabVIEW and .NET code modules, you must select the **LabVIEW Runtime** option in the LabVIEW Adapter Configuration dialog box so LabVIEW code modules execute in the same process as the TestStand Engine and the .NET code modules, which allows LabVIEW and .NET code modules to access the instrument sessions.

Debugging Considerations

Because you must use the **LabVIEW Runtime** option in the LabVIEW Adapter to share instrument sessions with .NET code modules, you cannot use the **Development System** option in the LabVIEW Adapter for debugging purposes in a system that shares instrument sessions between LabVIEW and .NET code modules.

If you use LabVIEW 2015 or later, you can attach the LabVIEW Development System to the TestStand Sequence Editor to debug LabVIEW VIs executing in the LabVIEW Run-Time Engine.

Using LabVIEW VI Analyzer (TSM)

Use the LabVIEW VI Analyzer tool to run tests that check VIs interactively for style, efficiency, and other aspects of LabVIEW programming, including the following tests, enabled by default, specific to TSM.

- **TSM Context Closing**—Verifies that a VI properly closes Semiconductor Module Context references. Detects cases where the output of a Semiconductor Module Context reference is not wired or not wired to a Close Reference function. Closing references in LabVIEW frees up memory that LabVIEW allocates for the references. Failure to close references causes reference leaks, which can negatively affect the performance of the VI over time.
- **Use High Resolution Polling Wait**—Verifies that VIs use the High Resolution Polling Wait VI instead of other wait functions, such as Wait (ms), Wait Until Next ms Multiple, or Time Delay. The High Resolution Polling Wait VI always waits for at least as long as the number of milliseconds specified, which makes it more precise than other wait functions.

Refer to the **LabVIEW Help** for more information about the LabVIEW VI Analyzer. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

Using LabVIEW Classes (TSM)

You cannot pass LabVIEW class references from LabVIEW code modules directly to TSM or from TSM to LabVIEW code modules. Use LabVIEW functional global variables instead of LabVIEW classes to share data between TSM and LabVIEW.

TSM Code Module API

Use the TSM Code Module API to develop code modules to perform tests using DUT pin or pin group names. You can use this API to manage pins, instruments, sessions, sites, data, switching, specifications, and to publish measurement data. The Semiconductor Multi Test and the Semiconductor Action steps create a `SemiconductorModuleContext` object that contains information about the pins and sites for the code module to test and stores the information on the `Step.SemiconductorModuleContext` property. Pass the `Step.SemiconductorModuleContext` property to the code modules to use the TSM Code Module API.

Using the TSM Code Module API in LabVIEW Applications

Use the TSM Code Module API VIs located on the **TestStand Semiconductor Module»Code Module Development** Functions palette.

Refer to the **TestStand Semiconductor Module Code Module API VIs Help**, located at `<LabVIEW>\help\lvtssemiconductor.chm`, for more information about the TSM Code Module API VIs. In LabVIEW, select **Help»LabVIEW Help** and navigate to the **VI and Function Reference»TestStand Semiconductor Module Code Module API VIs Help** book in the table of contents of the **LabVIEW Help** to launch the **TestStand Semiconductor Module Code Module API VIs Help**.

Using the TSM Code Module API in .NET Applications

Use the TSM Code Module API .NET class library to access the TSM Code Module API. Add a reference to the `<TestStand>\API\DotNET\Assemblies\CurrentV`

ersion\NationalInstruments.TestStand.SemiconductorModule.CodeModuleAPI.dll assembly to your Visual Studio project.

TestStand Semiconductor Module Code Module .NET API

April 2022, 375356J-01

This help file contains detailed information about the TestStand Semiconductor Module™ (TSM) Code Module .NET API.

Note If you open help files directly from the <TestStand>/Doc/Help directory, NI recommends that you open TSHelp.chm first because this file is a collection of all the TestStand help files and provides a complete table of contents and index.

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window.

Programming with TSM APIs in C#

In C#, you can access TSM APIs by adding the corresponding component as a reference in a project.

API	Component
TSM Code Module API	NationalInstruments.TestStand.SemiconductorModule.CodeModuleAPI
TSM Application API	NationalInstruments.TestStand.SemiconductorModule.ApplicationAPI

TSM installs .NET assemblies for the TSM Code Module and Application APIs in the <TestStand>\API\DotNet\Assemblies\CurrentVersion directory and in the Global Assembly Cache (GAC). The assemblies support .NET 4.0 and later.

To add a reference to TSM API assemblies in Visual Studio, select the project in the Solution Explorer and select **Project»Add Reference**. If the assemblies do not appear in the corresponding dialog box, exit all running copies of Visual Studio,

launch the TestStand Version Selector, select the current version of TestStand, and click the **Make Active** button.

Note If you have both 32-bit and 64-bit TestStand installed but only 32-bit TSM installed, TSM API assemblies do not appear in the Add References dialog box. Click the Browse tab and select the assembly from the `<TestStand>\API\DotNet\Assemblies\CurrentVersion` directory.

After you add one of the TSM API components as a reference in a project, add the following directive at the top of the source file:

```
using NationalInstruments.TestStand.SemiconductorModule;
```

or

```
using NationalInstruments.TestStand.SemiconductorModule.CodeModuleAPI;
```

All TSM classes, methods, and properties are available in the Microsoft Visual Studio object browser and are accessible from the C# source code. Help text for classes, methods, and properties appears in Microsoft IntelliSense.

To control NI instruments from a .NET assembly, you must also complete the following tasks:

- Install the appropriate NI driver for the instrument you want to control. Ensure you install the correct NI .NET API for the version of the NI driver you installed. For some NI drivers, you must manually select .NET support options in the installer.
- Add a reference to the corresponding NI driver .NET API assembly to the project. The Accelerometer with .NET and the Part Average Testing with .NET examples include Visual Studio projects that reference the TSM Code Module API and some NI .NET APIs and demonstrate how to invoke those APIs.

Visit `ni.com/info` and enter the Info Code `NETAPIdriversupport` for information about the available NI .NET APIs and the versions of the NI drivers each supports.

Upgrading TSM

(TSM 2019 or later) When you upgrade to TSM 2019 or later, you do not need to update .NET assembly references to the TSM Application API or TSM Code Module API assemblies in Visual Studio projects. The TSM installer ensures that references to earlier versions of the API assemblies are automatically redirected to the current version of the assembly.

(TSM 2017 or earlier) When you upgrade to a full, non-service pack version of TSM and you want to continue to use existing code modules and applications in the later version, you must update the .NET assembly references in Visual Studio projects. The assemblies you build using TSM APIs are compatible only with the version of TSM, including service packs, you reference in Visual Studio projects.

Upgrading Instrument Drivers

When you upgrade to a later version of an instrument driver .NET API, you do not need to update the .NET assembly references in Visual Studio projects. NI instrument driver .NET API installers ensure that references to earlier versions of the instrument driver .NET API assembly are automatically redirected to the current version of the assembly if the earlier version assembly does not exist.

The following table lists the minimum version of instrument driver .NET APIs TSM supports.

NI Driver	Assembly Name	Minimum Version
NI-DAQmx	NationalInstruments.DAQmx.dll	16.1
NI-DCPower	NationalInstruments.ModularInstruments.NIDCPower.Fx40.dll	1.1
NI-Digital Pattern	NationalInstruments.ModularInstruments.NIDigital.Fx40.dll	16.0
NI-DMM	NationalInstruments.ModularInstruments.NIDmm.Fx40.dll	15.2
NI-FGEN	NationalInstruments.ModularInstruments.NIFgen.Fx40.dll	16.0
NI-RFmx	NationalInstruments.RFmx.InstrMX.Fx40.dll	19.0

NI-RFPM	NationalInstruments.ModularInstruments.NIRfpm.Fx40.dll	14.5
NI-RFSA	NationalInstruments.ModularInstruments.NIRfsa.Fx40.dll	19.0
NI-RFSG	NationalInstruments.ModularInstruments.NIRfsg.Fx40.dll	19.0
NI-SCOPE	NationalInstruments.ModularInstruments.NIScope.Fx40.dll	2.0
NI-SWITCH	NationalInstruments.ModularInstruments.NISwitch.Fx40.dll	1.1

Exporting and Importing Test Limits with Text Files (TSM)

Use a tab-delimited text file to export and import test limits from and to Semiconductor Multi Test steps and Semiconductor Sequence Call steps in a single sequence file at edit time or run time. During development, export all the tests in a test program to review and edit and then import the changes back into the test program. At run time, load and execute a different set of test limits from separate text files based on the test program configuration you select by exporting the test limits and creating multiple copies of the file to edit for each unique set of test limits you want to use. To protect the test limits files from viewing or editing when deploying a test program, embed the test limits in the test program sequence file and use the TestStand password protection option to lock the sequence file.

Select **Semiconductor Module»Edit Test Program** and select **Test Limits Files** in the Test Program Editor to specify one or more test limits files to make available to the test program configurations. The test program configuration specifies the test limits file that loads before running a test lot.

Create a test limits file by selecting **Semiconductor Module»Export Test Limits from** or by clicking the **Export Test Limits from** button on the TSM toolbar to export test limits from a sequence file into a tab-delimited test limits text file. Import a test limits file by selecting **Semiconductor Module»Import Test Limits to** or by clicking the **Import Test Limits to** button on the TSM toolbar to import test limits from a tab-delimited test limits text file into a sequence file. When you import test limits from a text file, you can update limits in matching tests or replace all tests in matching steps.

Exporting Test Limits from Sequence Files (TSM)

Select **Semiconductor Module»Export Test Limits from <filename>** to export test limits from all test steps in a sequence file to a new or existing tab-delimited test limits text file at edit time. Selecting an existing file overwrites the content of the file.

When you export test limits, the text file includes the following test data for every sequence in the sequence file:

- **Sequence name**, **Step name**, **UniqueStepId**, **Test name**— Always included.
- **Test number**— If a step does not contain any tests, this field displays **No Tests**.

Note Use the Semiconductor Action step instead of the Semiconductor Multi Test step if the step does not contain any tests.

- **Pin or pin group**—TSM does not export the test data associated with the generated tests for the pins in the pin group.
- **Low limit expression**, **High limit expression**—For Pass/Fail tests, these fields are blank.
- **Scaling factor**—Included only when non-empty scaling factors exist for any test in the sequence file and scaling factors are the same within each test. The Units Prefix column of the Supported Scaling factors table lists values for exported scaling factors.
- **Low limit scaling factor**, **High limit scaling factor**, **Data limit scaling factor**—Included only when non-empty for any test in the sequence file and the data and limits scaling factors differ in at least one test in the sequence file. The **Units Prefix** column of the Supported Scaling Factors table lists values for exported scaling factors.
- **Evaluation comparison mode**—Included only when at least one step contains non-default value. For Pass/Fail tests, this field is blank.
- **Base units**—For Pass/Fail tests, this field is blank.
- **Evaluation type**, **Software bin**— Always included.

- **Test name expression**, **Test number expression**, **Software bin expression**, **Published data ID**, **Test data source expression**, **Export data to expression**—Included only when non-empty values exist for any tests in the sequence file.
- **Test numeric display format**—Included only when non-default test numeric display formats exist for any tests in the sequence file.
- **PAT base test number**—Included only when non-empty PAT base test numbers exist for any tests in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** and **Enable Static PAT** columns on the Semiconductor Multi Test Part Average Testing (PAT) tab do not include a checkmark for the test.
- **Enable dynamic PAT**—Included only when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab includes a checkmark in the sequence file.
- **Dynamic PAT test number**, **Dynamic PAT test name**, **Dynamic PAT software bin**, **Dynamic PAT low limit**, **Dynamic PAT high limit**—Included only when non-empty values exist for any tests in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.
- **Enable static PAT**—Included only when the **Enable Static PAT** column on the Semiconductor Multi Test Part Average Testing tab includes a checkmark in the sequence file.
- **Static PAT test number**, **Static PAT test name**, **Static PAT software bin**—Included only when non-empty values exist for any tests in the sequence file. A blank value is exported for a test when the **Enable Static PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.

The file does not contain test data for external sequence files referenced in the sequence file from which you export the test limits.

Notes

- For Semiconductor Multi Test steps in the sequence that have no tests, the file contains a row with the step information and the value `No Tests` for the Test Name. All other columns are blank. Use the Semiconductor Action step instead of the Semiconductor Multi Test step if the step does not contain any tests.
- For Semiconductor Sequence Call steps in the sequence that have no tests, the file contains a row with the step information and the value `No Tests` for the Test Name. All other columns are blank. Use the TestStand Sequence Call step instead of the Semiconductor Sequence Call step if the step does not contain any tests.
- If a field name begins with a mathematical sign, Microsoft Excel might interpret the contents of the cell as a formula and return an error for the cell. To work around this issue, format the cell in Excel as text data.
- In some cases when a limit has a large number of digits, Microsoft Excel might truncate the decimal portion of the number. To work around this issue, format the columns that contain the limit numbers in Excel as text data.
- Exporting PAT limits does not honor settings you specify in the `StepSettingsPaneUI` property of the `FileGlobals.PartAverageTestingAlgorithmDescription.EnvironmentSettings` container of the `PartAverageTestingCallbacks.seq` file.

See Also

Editing Test Limits Text Files

Opening Test Limits Text Files in Microsoft Excel

Test Limits Text File Structure (TSM)

Each row in the test limits file corresponds to a test. The import mode and the columns in the test limits file define how the Import/Export Test Limits tool matches rows to the sequence file.

Tags enclosed in angle brackets denote the properties recognized during import or export. The Import/Export Test Limits tool ignores any column with an unknown tag and any text above or on the same line as the opening `<SemiconductorModuleTests>` tag or below or on the same line as the required closing `</SemiconductorModuleTests>` tag. You can place comments above the `<SemiconductorModuleTests>` tag, below the `</SemiconductorModuleTests>` tag, or to the right of the recognized data columns, as shown in the following table:

Column Tag	Description	Format
`<SequenceName>`	Name of the sequence. The tool uses this column, if present, to match each row in the test limits file to locations in the sequence file.	String
`<StepName>`	Name of the step. The tool uses this column, if present, to match each row in the test limits file to locations in the sequence file.	String
`<StepId>`	Unique ID of the step automatically generated by TestStand. The tool uses this column, if present, to match each row in the test limits file to locations in the sequence file.	String
`<TestName>`	Name of the test.	String
`<TestNumber>`	Number of the test. Importing an empty string indicates the lack of a test number. When you enable the **Update limits in matching tests** option, the tool	Number

	uses this column, if present, to match each row in the test limits file to locations in the sequence file. If a step does not contain any tests, this field displays `No Tests`. **Note** Use the Semiconductor Action step instead of the Semiconductor Multi Test step if the step does not contain any tests.	
`<Pin>`	Name of the pin or pin group. For a pin group, test data for individual pins in the pin group are not included in the file.	String
`<LowLimitExpression>`	Lower limit expression. Blank for Pass/Fail tests.	String
`<HighLimitExpression>`	Upper limit expression. Blank for Pass/Fail tests.	String
`<ScalingFactor>`	Scaling factor used to display the data, high limit, and low limit fields. The **Units Prefix** column of the Supported Scaling Factors table lists valid values for scaling factors. **Note** If you use the <ScalingFactor> column, you cannot use the individual field <LowLimitScalingFactor>, <HighLimitScalingFactor	String

	>, and <DataScalingFactor> columns.	
<LowLimitScalingFactor>	Scaling factor used to display the low limit field. The **Units Prefix** column of the Supported Scaling Factors table lists valid values for scaling factors.	String
<HighLimitScalingFactor>	Scaling factor used to display the high limit field. The **Units Prefix** column of the Supported Scaling Factors table lists valid values for scaling factors.	String
<DataScalingFactor>	Scaling factor used to display the data field. The **Units Prefix** column of the Supported Scaling Factors table lists valid values for scaling factors.	String
<ComparisonType>	Evaluation Comparison Mode. Included in file only when property contains non-default value. Blank for Pass/Fail tests.	String
<Units>	Base units of the limits. Blank for Pass/Fail tests.	String
<EvaluationType>	Evaluation type. Valid values are Pass/Fail or Numeric Limit.	String
<FailBin>	Software Bin.	Number
<TestNameExpression>	Test name expression. Included in file only when non-empty test name expressions exist in the sequence file.	String
<TestNumberExpression>	Test number expression. Included in file only when non-empty test number expressions exist in sequence file.	String
<FailBinExpression>	Software bin expression. Included in file only when non-empty	String

	software bin expressions exist in sequence file.	
<PublishedDataId>	Published data ID. Included in file only when non-empty published data IDs exist in the sequence file.	String
<TestDataSource>	Test data source expression. Included in file only when non-empty test data source expressions exist in the sequence file.	String
<ExportLocation>	Expression for location to which to export data (Export Data To column in the Tests table). Included in file only when non-empty export data to expressions exist in the sequence file.	String
<NumericFormat>	Test numeric display format that applies to the high limit, low limit, and data. During export, TSM retrieves the value from the `Data` property of the test. Included in file only when non-default numeric formats exist for any tests in the sequence file.	String
<PAT_BaseTestNumber>	Base test number for PAT tests. Included in file only when non-empty PAT base test numbers exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** and **Enable Static PAT** columns on the Semiconductor Multi Test Part Average Testing (PAT) tab do not include a checkmark for the test.	Number
<DynamicPAT_Enabled>	Enables dynamic PAT. Included in file only when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Ave	Boolean

	rage Testing tab includes a checkmark in the sequence file.	
<DynamicPAT_TestNumber>	Number of the dynamic PAT test. Included in file only when non-empty dynamic PAT test numbers exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	Number
<DynamicPAT_TestName>	Name of the dynamic PAT test. Included in file only when non-empty dynamic PAT test names exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	String
<DynamicPAT_FailBin>	Dynamic PAT software bin. Included in file only when non-empty dynamic PAT software bins exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	Number
<DynamicPAT_LowLimit>	Dynamic PAT lower limit. Included in file only when non-empty dynamic PAT low limits exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does	Number

	not include a checkmark for the test.	
<DynamicPAT_HighLimit>	Dynamic PAT higher limit. Included in file only when non-empty dynamic PAT high limits exist in the sequence file. A blank value is exported for a test when the **Enable Dynamic PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	Number
<StaticPAT_Enabled>	Enables static PAT. Included in file only when the **Enable Static PAT** column on the Semiconductor Multi Test Part Average Testing tab includes a checkmark in the sequence file.	Boolean
<StaticPAT_TestNumber>	Number of the static PAT test. Included in file only when non-empty static PAT test numbers in the sequence file. A blank value is exported for a test when the **Enable Static PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	Number
<StaticPAT_TestName>	Name of the static PAT test. Included in file only when non-empty static PAT test names exist in the sequence file. A blank value is exported for a test when the **Enable Static PAT** column on the Semiconductor Multi Test Part Average Testing tab does not include a checkmark for the test.	String
<StaticPAT_FailBin>	Static PAT software bin. Included in file only when non-empty s	Number

	tatic PAT software bins exist in t he sequence file. A blank value i s exported for a test when the **E nable Static PAT** column on t he Semiconductor Multi Test Pa rt Average Testing tab does not include a checkmark for the tes t.	

Note Exporting PAT limits does not honor settings you specify in the `StepSettingsPaneUI` property of the `FileGlobals.PartAverageTestingAlgorithmDescription.EnvironmentSettings` container of the `PartAverageTestingCallbacks.seq` file.

See Also

Editing Test Limits Text Files

Opening Test Limits Text Files in Microsoft Excel

Opening Test Limits Text Files in Microsoft Excel (TSM)

Complete the following steps to open a test limits text file in Microsoft Excel.

1. In Excel, select **Open**. Select **All Files** in the drop-down menu of file types.
2. Select the test limits file you want to open and click the **Open** button.
3. In the Text Import Wizard, select **Delimited** in the **Original data type** option and click **Next**.
4. Select **Tab** in the **Delimiters** option and click **Next**.
5. Select **General** in the **Column data format** option and click **Finish**.

Alternatively, you can drag and drop a test limits text file into Excel to open the file.

When you save a modified test limits file that you opened or edited in Excel, Excel might return a message that the Excel file contains features that are not compatible with the tab-delimited text format. Click **Yes** in the prompt to save the workbook in

text format. The Import/Export Test Limits tool is unable to parse any additional formatting information Excel adds to the file.

Notes

- If a field name begins with a mathematical sign, Microsoft Excel might interpret the contents of the cell as a formula and return an error for the cell. To work around this issue, format the cell in Excel as text data.
- In some cases when a limit has a large number of digits, Microsoft Excel might truncate the decimal portion of the number. To work around this issue, format the columns that contain the limit numbers in Excel as text data.

Editing Test Limits Text Files (TSM)

You can delete columns in the test limits text file to simplify the external limits file, but if you delete an identifier column, such as SequenceName, StepName, StepId, or TestName, the text file might then contain rows that use the same identifier string. When this situation occurs, you must consolidate duplicate rows or include another identifier to distinguish among tests. You must repeat the customizations you make to the text file each time you export the test limits.

For example, if you have several steps or tests in a sequence file with the same name or test number but you want to use the same limits for each step, you can delete the StepId column and possibly the SequenceName column to simplify the resulting text file. However, if you want to use different limits for steps or tests with the same name or test number, the StepId column and possibly the SequenceName column must remain in the text file to distinguish the tests.

Notes

- If a field name begins with a mathematical sign, Microsoft Excel might interpret the contents of the cell as a formula and return an error for the cell. To

work around this issue, format the cell in Excel as text data.

- In some cases when a limit has a large number of digits, Microsoft Excel might truncate the decimal portion of the number. To work around this issue, format the columns that contain the limit numbers in Excel as text data.

See Also

Opening Test Limits Text Files in Microsoft Excel

Test Limits Text File Structure

Importing Test Limits from Text Files (TSM)

Select **Semiconductor Module»Import Test Limits into <filename>** to import test limits from a tab-delimited test limits text file into all test steps in a sequence file at edit time. When you import test limits from a text file, you can replace only the tests defined in the test limits file or you can delete all the tests in the sequence file and recreate each test from the contents of the test limits file.

The Import/Export Test Limits tool uses the SequenceName, StepName, and StepId columns in the test limits file to match tests in the sequence file. When you update the limits in matching tests from the test limits file, the tool also uses the TestNumber column to identify the test. If one of these identifier columns does not exist in the test limits file, the tool imports data into any step or test that matches the remaining identifiers and a row in the test limits file might update multiple locations in the sequence file. If more than one location in the sequence file matches the identifiers in a row in the test limits file, TSM imports data from that row into all the matching locations.

The Import/Export Test Limits tool returns an error and fails to import a test limits file if:

- A row in the test limits file does not match any location in the sequence file.
- A single test in the sequence file matches more than one row in the test limits file.

- A location in the sequence file does not match any row in the test limits file and the **Require every step to be in test limits file** option is enabled.
- A row in the test limits file matches a Semiconductor Sequence Call step in the sequence file and the Test Data Source column contains a value.

The Import Test Limits into Sequence File dialog box contains the following options:

- **Update limits in matching tests**—Finds any test in the sequence file that matches the SequenceName, StepName, StepId, and TestNumber identifier columns in the test limits file and sets the property values in the sequence file to the corresponding values in the test limits file. If one of these identifier columns other than TestNumber does not exist in the test limits file, the tool sets the corresponding property values for any step that matches the remaining identifiers.

Note You cannot change the test number for tests in the sequence file when you select this mode. You must use the **Replace all tests in matching steps** option if you want to change the test number.

The tool modifies only the tests and properties that exist in the test limits file. Tests and properties defined in the sequence file but not in the test limits file remain in the sequence file unchanged.

Select this option when you want to set only certain test properties at run time.

- **Replace all tests in matching steps**—Deletes all the tests that correspond to steps in the test limits file and recreates each test from the contents of the test limits file by adding tests to any step that matches the SequenceName, StepName, and StepId identifier columns in the test limits file. If one of these identifier columns does not exist in the test limits file, the tool adds a test to any step that matches the remaining identifiers. For columns that do not exist in the test limits file, the tool sets the corresponding property to the default value in the sequence file. The test limits file must include the columns in which you use non-default values; otherwise, the tool

uses default values for those properties in the sequence file. If any step in the test limits file has a value of `No Tests` for the Test Number, the tool deletes all tests for any matching steps.

Note Use the Semiconductor Action step instead of the Semiconductor Multi Test step if the step does not contain any tests.

Select this option when you want to use limit files with different numbers of tests. For example, you might want to include more tests in a normal testing mode than you include in a QA testing mode.

- **Require every step to be in test limits file**—Returns an error and does not import the file if a test or step exists in the sequence but does not exist in the limits file. If you do not enable this option, the Import/Export Test Limits tool returns a warning and imports only matching tests or steps in the limits file.

See Also

Tutorial: Importing Test Limits from a File

Exporting a Correlation Offsets Template File (TSM)

You can apply correlation offset values to test results on a per-site basis at run time before evaluating the test result data against limits. Use the Load Correlation Offsets Step and associated edit tab to load and apply a correlation offset file.

Select **Semiconductor Module»Export Correlation Offset Template file based on <filename>** to generate a tab-delimited correlation offsets template file (`.txt`) based on the numerical limit tests in the selected sequence file. If you make changes, export the file again, and select an existing file, the content of the existing file is overwritten, including any custom correlation offset values you set.

The correlation offsets template file contains entries (`0.0`) for all numerical limits tests for all defined sites in the sequence file. You can use the template file as a starting point for a custom correlation offsets file.

The file does not contain correlation offsets entries for external sequence files referenced in the sequence file from which you export the correlation offsets template file.

The correlation offsets template file contains the following test data for every sequence in the sequence file:

Data Item	Description
SequenceName	Always included in the file.
StepName	Always included in the file.
StepId	Always included in the file.
TestNumber	Always included in the file. If a step does not contain any tests, this field displays `"No Tests"`.
Site_0_Offset to Site_**n**_Offset	**n** is the number of sites defined for the station. If no sites are defined, only `Site_0_Offset` will exist. The default value is `0.0`. You do not have to assign a custom correlation offset value to every test in the file.

Debugging TestStand Test Programs

Consider the following common approaches to debug the following components of a TestStand test program.

Sequence Execution Code Modules and Related Files Analysis and Rule Configuration Modules Memory Management Deployments

Sequence Execution

Sequences can be in a running or suspended testing state. Testing is running when continuously testing DUTs. When testing is running, each test socket is executing tests for one DUT. Testing is suspended when one or more TestStand test socket executions are suspended at a breakpoint. One test socket can be suspended while other test sockets are running. The background of the Execution window changes to yellow to indicate that the execution is suspended. You can examine sequence variables only in executions that are suspended.

Use the following techniques to debug sequence execution:

- **Set Sequence Breakpoints**—To suspend testing at a specific step, set a breakpoint on the step by clicking in the column to the left of the step name in the Steps Pane of the Sequence File window or in the Execution window. You can right-click the breakpoint stop sign icon and select **Breakpoint»Breakpoint Settings** to launch the Breakpoint Settings dialog box, in which you can specify an expression to set a conditional breakpoint. Testing suspends when the execution reaches the step breakpoint. A yellow arrow icon, called the execution pointer, to the left of the step indicates the next step to execute when testing resumes. Click the breakpoint icon to remove the breakpoint.
- **Set Data Breakpoints**—To suspend testing when the value of a variable changes or when it has a specific value, use the Watch View pane to enter an expression for the variable and edit the watch expression to specify a break condition. TestStand evaluates these break conditions when each step executes and suspends the testing if the break conditions are met.

Note Delete watch expressions when you no longer need them because they can negatively affect execution performance.

- **Control Test Program and Step Execution**—When testing is suspended, use the following options in the Debug menu or toolbar or in the Steps pane context menu of the Execution window to control test program and step execution:
  - **Debug Menu/Toolbar**
    - **Step Into**—Enters and suspends inside the code module associated with the step to which the execution pointer points.
    - **Step Over** —Executes the step to which the execution pointer points and suspends execution on the next step in the sequence.
    - **Step Out**—Resumes execution through the end of the current sequence and suspends execution on the next step in the calling sequence.
  - **Steps Pane Context Menu**

- **Run Mode»Skip** (temporarily skips step)—Select **Skip** to prevent the step from executing in the current execution. Changing the step run mode in an execution does not persist the setting to the step in the sequence file. As a result, once execution ends, the setting reverts to the original run mode before you made the change. Change the run mode on the step in the Sequence File window to persist the setting in the sequence file.
- **Set Next Step to Cursor** —Moves the execution pointer to point to a different step to change which step executes next.
- **Run Selected Steps**—Select one or more arbitrary steps in the sequence that you want to execute before resuming normal execution. TestStand executes the selected steps in the order they appear in the sequence. If a breakpoint suspends execution of the selected steps, the Steps pane displays the interactive execution pointer, which differs from the normal execution pointer.

- **View and Modify Variable and Expression Values** —Use the following techniques to view or modify variable values and to monitor expression values:
  - When testing is suspended, use the Variables pane to view and modify the values of any of the variables and properties accessible in the current execution.
  - To monitor the value of an expression while testing, enter the expression in the Watch View pane when testing is suspended. You can also use the Edit Breakpoints/Watch Expression dialog box to enter expressions when testing is not in progress. When you enable tracing, the sequence editor updates the values after each step executes.
- **Display Debug Messages**—To display messages to the Output pane while testing, use the OutputMessage expression function in a Statement step.
- **Use Execution Profiler**—The Execution Profiler records event, operation, item, thread, and execution data, such as the duration of steps, code modules, and other resources a multithreaded TestStand system uses over a period of time.

Code Modules and Related Files

The TestStand Sequence Editor integrates with common code module development environments to help streamline debugging tasks.

LabVIEW

When executing VIs in the LabVIEW Development System, you can debug the VI code modules loaded in memory using the same version of LabVIEW in which the VIs are executing by setting breakpoints and probes in the VIs. Once the TestStand execution reaches the VI, the first breakpoint is triggered in a new instance of the VI. To add additional breakpoints or probes, add them to this instance of the VI. To make changes to the VI after debugging, right-click the front panel of the debugging instance of the VI and select **Remote Debugging»Quit Debug Session**.

The LabVIEW Development System can also debug applications that execute VIs using the LabVIEW Runtime, such as VIs executed using the TestStand LabVIEW Adapter and LabVIEW-built DLLs executed using the TestStand C/C++ DLL Adapter.

When the TestStand LabVIEW Adapter loads a VI code module, it reserves the VI in LabVIEW to prevent edits to the VI. To edit a reserved VI, you must first unload the step code module. Click the Unload and Open VI button on the LabVIEW Module tab to unload a LabVIEW step code module and open the VI for editing. The button is enabled only if all executions are suspended. If the button tooltip indicates that it is not enabled because there are executions that are actively running, use the Break All button on the Debug toolbar to suspend all executions.

You can also use the LabVIEW Desktop Execution Trace Toolkit to collect various types of trace data from LabVIEW applications.

Packed Libraries

LabVIEW packed libraries, or `.lvlibp` files, contain compiled LabVIEW code that you can execute from TestStand. By default, VIs in packed libraries do not contain any source code and cannot be debugged. To create a packed library that you can debug, select the **Enable Debugging** option in the Advanced page of the Packed Library Properties dialog box. You can debug VIs in a debuggable packed library the same way you debug stand-alone VIs when you execute the VIs using the

Development System option in the TestStand LabVIEW Adapter Configuration dialog box.

LabVIEW, LabWindows/CVI, and .NET DLLs

To debug a DLL TestStand calls, first create the DLL with debugging enabled in the application development environment (ADE). Then, launch the TestStand Sequence Editor or TestStand User Interface executable from the ADE or attach to the sequence editor or user interface process from the ADE, when supported.

.NET

To debug a .NET assembly, first create the assembly with debugging enabled in the ADE. Then, launch the sequence editor or user interface from Microsoft Visual Studio or attach to the sequence editor or user interface process from Visual Studio.

You cannot view any TestStand sequence variables while suspended at a breakpoint in Visual Studio.

You cannot modify .NET assemblies while TestStand has the assemblies loaded. Before you can successfully rebuild a .NET assembly loaded by TestStand, you must unload the assembly and all other assemblies the .NET Adapter is using in TestStand by completing the following steps:

1. End all executions.
2. Clear any references to .NET objects stored in station globals.
3. Select **File»Unload All Modules**.

To unload .NET code modules, TestStand must unload the .NET appdomain that it uses with the .NET Adapter and can do so safely only when the .NET appdomain is no longer in use.

Analysis and Rule Configuration Modules

If you use the TestStand Sequence Analyzer, you can debug custom analysis modules and rule configuration modules by setting a breakpoint in the module and then calling the module from the TestStand Sequence Analyzer.

Memory Management

Visit `ni.com/info` and enter the Info Code `tsmemory` to access the NI tutorial, Troubleshooting Memory Growth Issues in TestStand Systems, for information about efficiently managing memory in test sequences and code modules.

Deployments

Ensure that you have followed the suggested deployment process to plan, design, implement, execute, and validate a full or patch deployment for a TestStand-based test system.

The TestStand Deployment Utility generates several errors and warnings specifically related to debugging patch deployments. Additionally, you can use the Build Status tab of the deployment utility to view information about the patch deployment.

Debugging TSM Test Programs

In addition to following common approaches to debug components of a TestStand test program, consider the following additional approaches to debug components of a TSM test program

Sequence Execution Lot Statistics Viewer Simulated Handler or Prober

Sequence Execution

Sequences can be in a running or suspended testing state. Testing is running when continuously testing DUTs. When testing is running, each test socket is executing tests for one DUT. Testing is suspended when one or more TestStand test socket executions are suspended at a breakpoint. One test socket can be suspended while other test sockets are running. The background of the Execution window changes to yellow to indicate that the execution is suspended. You can examine sequence variables only in executions that are suspended.

Pausing a lot does not suspend executions the way using the default TestStand Sequence Editor **Debug»Break** or **Break All** menu items do. When you pause a lot with the **Pause** button, testing continues until each site completes testing its current DUT. The executions then wait for operator input before sending the End of Test signal to the handler or prober and proceeding with testing the next batch of

DUTs. To suspend test socket executions to examine variables or to step into code modules, set a breakpoint on the step where you want to suspend. You cannot examine variables in the sequence editor for executions that are not suspended, even if they are paused between DUTs.

Use the following techniques to debug TSM sequence execution:

- **Use TSM Toolbar** —Use the TSM toolbar buttons to control execution and view lot statistics while debugging a sequence. Avoid using the default TestStand Sequence Editor **Debug»Terminate**, **Terminate All**, **Abort**, and **Abort All** menu items because they might not clean up and close instrument sessions correctly.
- **Step Into TSM Steps**—For Semiconductor Multi Test and Semiconductor Action steps that execute test code for multiple sites, the test sockets for all the sites that the step is testing must be suspended at the same step before you can use the Step Into Debug menu item or toolbar button.

  For LabVIEW 2020 and newer, when you step into a VI from TestStand and click **Run**, TSM opens the block diagram, suspends execution on the first node, and automatically enables **Retain Wire Values**. You can also edit the executing VI in LabVIEW and continue execution of the lot in TestStand.

Lot Statistics Viewer

To monitor the bin counts for each site while testing in the sequence editor, use the Lot Statistics Viewer, in which you can view lot statistics while running or debugging a sequence in the sequence editor. You can also control test program execution in the Lot Statistics Viewer.

Simulated Handler or Prober

When developing, testing, or debugging a semiconductor test program, the test developer might not have access to a real handler or prober. In such cases, the test developer can use the NI Built-in Simulated Handler Driver to verify that the test program behaves correctly without a handler or prober.

See Also

Semiconductor Module Run-Time Error Dialog Box

Editing an Executing VI in LabVIEW

For LabVIEW 2020 and newer, when you step into a VI from a TestStand step, LabVIEW displays the **Edit VI and reset step execution** button on both the front panel and block diagram toolbars. Use this button to stop executing the VI and to open it for editing. When you click **Edit VI and reset step execution**, TestStand resets the step execution to the step that you stepped into. Continuing execution of the lot in TestStand will execute the VI with the modifications you made.

Debugging RF Sessions (TSM)

You can enable debugging for RF instruments so that you can use an NI-RFmx soft front panel to debug opened sessions for the instruments. Because enabling debugging for RF instruments might incur a performance penalty, TSM automatically disables debugging for VST and RFSA instruments on the system when you run a test program in an operator interface. TSM automatically enables debugging for those instruments when you run a test program in the sequence editor so you can use a soft front panel to debug NI-RFmx sessions while running in the sequence editor.

To prevent TSM from automatically changing the debugging option for RF instruments, change the file extension on the `<TestStand>\Components\Models\ModelPlugins\Addons\NI_SemiconductorModule_RFmxDebugToggle.seq` file to something other than `.seq`.

Custom Instrument Panels (TSM)

You can use custom pin- and site-aware instrument panel VIs to control and measure pins during test program execution at a breakpoint, which can be useful during test program development and troubleshooting. Custom instrument panels obtain active instrument sessions stored in Semiconductor Module context objects using the Set Session VIs or .NET methods in TSM Code Module API. During active test program execution, TSM disables the custom instrument panel to avoid conflicts.

Creating Custom Instrument Panels

You must create the following components to implement a custom instrument panel you can launch from the Semiconductor Module menu:

- A custom instrument panel callback sequence file that meets the following requirements:
  - The file must reside in the <TestStand Public>\Components\Modules\NI_SemiconductorModule\CustomInstrumentPanels directory.

Note After you add custom instrument panel components to this directory, you must restart TestStand to register the custom instrument panel and update the **Semiconductor Module»Custom Instrument Panels** menu.

  - The file must contain a `LaunchCustomInstrumentPanel` callback sequence that includes a TestStand Action step to launch the custom instrument panel VI. Ensure that you enable the **Show VI Front Panel** option on the Module tab of the step. The `LaunchCustomInstrumentPanel` callback sequence must also include a Parameters variable of type Object Reference named `SemiconductorModuleManager`.
  - The file must include a file global variable named `CustomInstrumentPanelDisplayName`. This variable must include a unique value to display in the **Semiconductor Module»Custom Instrument Panels** menu.
  - The file can include an optional file global variable named `CustomInstrumentPanelIcon` to specify the filename of an icon to display in the **Semiconductor Module»Custom Instrument Panels** menu. The icon must be located in the `C:\Users\Public\Documents\National Instruments\<TestStand>\Components\Icons` directory.
- A custom instrument panel VI, for example, `TopLevel.vi`, that implements debugging tasks for the instrument and that meets the following requirements:

- You must call the VI from a TestStand Action step in the LaunchCustomInstrumentPanel callback sequence.
- You must use the `Parameters.SemiconductorModuleManager` string to pass the Semiconductor Module Manager object reference to the VI.

Note NI recommends that custom instrument panel VI source code resides in the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\CustomInstrumentPanels` directory along with the required custom instrument panel callback sequence file.

Refer to the following resources that you can use as starting points for custom instrument panels you create:

- Examples located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instrument Panels` directory
- A template for the required custom instrument panel callback sequence file named CustomInstrumentPanel.seq located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory
- A template for the required corresponding custom instrument panel LabVIEW project files located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates\CustomInstrumentPanelTemplate` directory

Note To modify the installed examples or templates, copy the files from the existing locations to the `<TestStand Public>` directory and make changes to the copies of the files.

Launching Custom Instrument Panels

Select **Semiconductor Module»Custom Instrument Panels** and select from the available options to launch the corresponding custom instrument panel VI. Click the **Close (X)** button on the custom instrument panel VI titlebar to end the execution of the VI.

Although you can launch a custom instrument panel at any time, the VI will not have access to a valid Semiconductor Module context object reference for pin- and site-aware instrument sessions until all test program executions are at a breakpoint. Additionally, the Semiconductor Module context object reference is not available to custom instrument panel VIs when any test program execution is at a breakpoint in the ProcessCleanup callback.

Viewing Multisite Data in Code Modules (TSM)

Often you cannot directly examine multisite data by probing a LabVIEW wire or viewing the value of a .NET variable because typical TSM test code modules produce data organized in an array indexed by instrument session instead of by site.

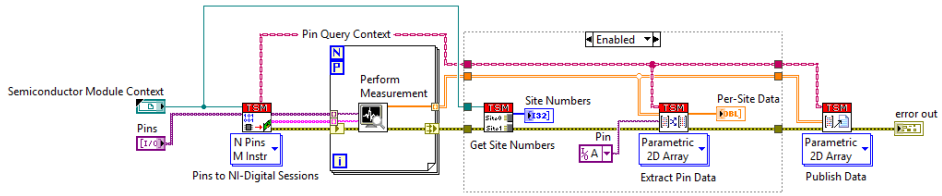
To view multisite data in code modules while debugging, use the following TSM VIs or .NET methods in your code module to convert per-instrument data to per-site data:

- **Extract Pin Data VI and ExtractPinData .NET method**—Extract measurement data for a single pin in the pin query from per-instrument measurement data and return per-site data.
- **Per-Instrument to Per-Site Data VI and PerInstrumentToPerSiteData .NET method**—Convert per-instrument measurement data to per-site data for all pins in the pin query.

Note TSM orders per-site data in the same order as the sites returned by the Get Site Numbers VI or SiteNumbers .NET property.

Extracting Per-Site Data for A Single Pin

LabVIEW



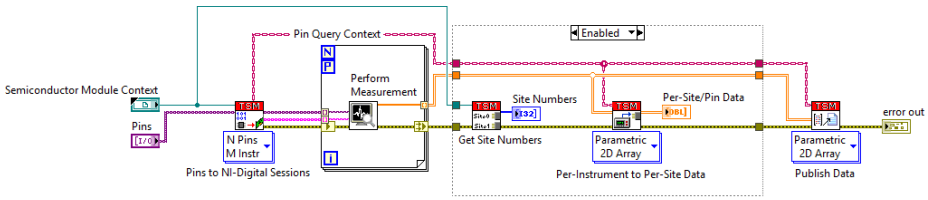
The code inside the diagram disable structure obtains per-site data for pin A.

.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string[] pins)
{

var pinQueryContext =
semiconductorModuleContext.GetNIDigitalPatternSessionsForPpmu(pins, out var
digitalSessions, out var pinSetStrings);
double[][] measurements = PerformMeasurement(digitalSessions, pinSetStrings);

#if DEBUG

var siteNumbers = semiconductorModuleContext.SiteNumbers;

// Extract per-site data for a single pin for viewing in the debugger. The
perSiteMeasurementsForAllPins variable is
// an array of measurements. Each element is the measurement for pin A on a
specific site. For example,
// perSiteMeasurementsForPinA[1] contains the measurement for pin A on the site
specified by siteNumbers[1].

var perSiteMeasurementsForPinA =
pinQueryContext.ExtractPinData(measurements, "A");

#endif

pinQueryContext.Publish(measurements);

}
```

Extracting Per-Site Data for All Pins

LabVIEW



The code inside the diagram disable structure obtains per-site data for or all pins.

.NET (C#)

```
public static void ExampleCodeModule(ISemiconductorModuleContext
semiconductorModuleContext, string[] pins)
{

var pinQueryContext =
semiconductorModuleContext.GetNIDigitalPatternSessionsForPpmu(pins, out var
digitalSessions, out var pinSetStrings);
double[][] measurements = PerformMeasurement(digitalSessions, pinSetStrings);

#if DEBUG

var siteNumbers = semiconductorModuleContext.SiteNumbers;

// Convert per-instrument data to per-site data for viewing in the debugger. The
perSiteMeasurementsForAllPins variable is
// an array of arrays. Each element is an array of measurements for all pins on a
specific site. For example,
// perSiteMeasurements[1][0] contains the measurement for the pin specified by
pins[0] on the site specified by siteNumbers[1].

var perSiteMeasurementsForAllPins =
pinQueryContext.PerInstrumentToPerSiteData(measurements);

#endif

pinQueryContext.Publish(measurements);

}
```

See Also

Organization of Measurement Data

Using Environments in TSM

You can configure TSM-specific environments. TSM is enabled by default in custom TSM environments you create. You can disable and re-enable TSM in a custom TSM environment. Use the Configure Environment dialog box to create, load, and edit an environment.

Offline Mode (TSM)

Use Offline Mode to develop, run, and debug test programs only on a computer without access to NI instruments. Offline Mode simulates the NI instruments the test program needs. Using Offline Mode does not require changes to the test program. Use the Accelerometer example to explore Offline Mode.

Using Offline Mode can be helpful when multiple people are developing a test program at the same time and must share access to a tester, when you do not have access to a tester, or when you want to explore different test program approaches that require different instruments.

See Also

Offline Mode Requirements

Offline Mode Workflow

Offline Mode Requirements (TSM)

Notes

- You must have a valid TSM license to use Offline Mode.
- STS Software 19.0 or later includes the required software versions to use Offline Mode.

Offline Mode requires NI DAQmx 19.1 or later and PXI Platform Services 19.1 or later. Additionally, you must install the following instrument drivers to simulate the corresponding instruments:

- NI-DCPower 19.1 or later
- NI-Digital Pattern Driver 19.0 or later
- NI-DMM 19.1 or later
- NI-FGEN 19.1 or later
- NI-HSDIO 19.0 or later

- NI-RFPM 19.0 or later, including IVI shared components
- NI-RFSA 19.1 or later
- NI-RFSG 19.1 or later
- NI-SCOPE 19.1 or later
- NI-SWITCH 19.1 or later
- NI-Sync 19.0 or later

See Also

Offline Mode Workflow

Offline Mode Workflow (TSM)

Complete the following steps to develop, run, and debug test programs only on a computer without access to NI instruments.

1. Install STS Software 19.0 or later or ensure you meet the requirements on the computer on which you want to use Offline Mode.
2. If necessary, copy the test program and supporting files from the STS you want to simulate in Offline Mode to the computer on which you want to use Offline Mode.
3. Obtain an Offline Mode system configuration file.
4. Copy the Offline Mode system configuration file to a `Supporting Materials\Offline Mode Configurations\`**subdirectory** located in the same directory as the test program on the computer on which you want to use Offline Mode.
5. On the computer on which you want to use Offline Mode, open the test program you want to execute in Offline Mode.
6. Launch the Test Program Editor, navigate to the Offline Mode panel, and specify the Offline Mode system configuration file to associate with the test program.
7. Enable Offline Mode.
8. Execute the test program.
9. Debug the test program.

10. Disable Offline Mode.
11. If necessary, copy the test program and supporting files from the computer on which you are using Offline Mode to the STS.

Obtaining an Offline Mode System Configuration File (TSM)

The Offline Mode system configuration file defines the instruments on a specific STS.

On the STS you want to simulate in Offline Mode, complete the following steps to export the Offline Mode system configuration file.

1. Launch STS Maintenance Software.
2. Select **File»Export Offline Mode System Configuration**.
3. Save the Offline Mode system configuration file in a `Supporting Materials\Offline Mode Configurations\`**subdirectory** located in the same directory as the test program.

If you do not have access to the STS you want to simulate in Offline Mode, use one of the following methods to obtain the Offline Mode system configuration file.

- Copy the TSM Accelerometer example Offline Mode system configuration file (`OfflineModeSystemConfigurationForAccelerometerExample.offlinecfg`) and modify the file to include the instruments on the STS you want to simulate in Offline Mode.
- Copy and modify an existing Offline Mode system configuration file to include the instruments on the STS you want to simulate in Offline Mode.
- Use the Offline Mode system configuration file XML structure to manually create the file.
- Contact NI to obtain the Offline Mode system configuration file for the specific STS you want to simulate in Offline Mode.

See Also

Offline Mode Workflow

Enabling Offline Mode (TSM)

Complete the following steps to enable Offline Mode on a computer without access to NI instruments. Offline Mode remains enabled, including after you exit and relaunch TSM, until you disable it.

Note You cannot enable Offline Mode on an STS or from a TSM Operator Interface.

1. Install STS Software 19.0 or later or ensure you meet the requirements on the computer on which you want to use Offline Mode.
2. Obtain an Offline Mode system configuration file and associate it with the test program you want to execute in Offline Mode. If you do not associate an Offline Mode system configuration file with the test program, TSM prompts you for an Offline Mode system configuration file when you enable Offline Mode, but the TSM Operator Interface does not.
3. Select **Semiconductor Module»Enable Offline Mode** or click the **Enable Offline Mode** button on the TSM toolbar to enable Offline Mode. The **Enable Offline Mode** button icon changes to indicate the current state of Offline Mode and the main menu bar displays an **Offline Mode** indicator to the right of the **Help** menu. Additionally, the `OfflineMode` property of the NI_SemiconductorModule_StandardStationSettings data type indicates the current state of Offline Mode. You can also use the Get Offline Mode VI or the `IsSemiconductorModuleInOfflineMode` .NET method to check the current status of the Offline Mode setting.

When you enable Offline Mode, TSM creates simulated versions of all instruments defined in the Offline Mode system configuration file. The simulated instruments persist until you disable Offline Mode or execute a test program with a different Offline Mode system configuration file. TSM also renames all physical instruments that conflict with the instruments defined in the Offline Mode system configuration file with an `NI_Offline_` prefix to avoid conflicts with simulated instruments and to prevent accidental damage to real instruments. Disabling Offline Mode restores the instrument configuration.

Note To avoid damage to real instruments, do not use simulated instruments and real instruments at the same time.

Enabling Offline Mode might fail when the computer on which you want to use Offline Mode does not include a required instrument driver, an instrument driver does not support simulation for an instrument, or an error exists in the Offline Mode system configuration file. Use the following techniques to resolve the issue:

- Install any missing drivers.
- Modify the test program to remove dependencies on unsupported instruments.
- Fix any errors in the Offline Mode system configuration file.

See Also

Offline Mode Workflow

Executing Test Programs in Offline Mode (TSM)

Note Although simulated instruments behave as closely as possible to real instruments, measurements and data differ in Offline Mode. Additionally, custom code that processes results, test time performance, instrument functionality, and error reporting might differ in Offline Mode and can result in different behavior of the test program.

Complete the following steps to execute a test program in Offline Mode on a computer without access to NI instruments.

1. Confirm that Offline Mode is enabled.
2. Click the **Start Lot** button or the **Single Test** button on the TSM toolbar.

Note TSM and the TSM Operator Interface update the simulated instruments to match the Offline Mode system configuration file associated with the test program before the test program begins execution. TSM does

not automatically update the simulated instruments when you run test programs any other way.

3. Review the results.

See Also

Offline Mode Workflow

Measuring and Publishing Values in Offline Mode (TSM)

Note Although simulated instruments behave as closely as possible to real instruments, measurements and data differ in Offline Mode. Additionally, custom code that processes results, test time performance, instrument functionality, and error reporting might differ in Offline Mode and can result in different behavior of the test program.

By default, TSM always uses data that cause all tests to pass when you run in Offline Mode. For Numeric Limit tests with limits, TSM uses the average of the limits. For Numeric Limit tests without limits, TSM logs `0`. For Pass/Fail tests, TSM uses the value `True`.

You can also use the published value for the tests or specify a value for tests you run in Offline Mode.

Set the following properties to use the published value for the tests.

- **Numeric Limit Test**—Set the `Step.Result.Evaluations[i].NumericLimit.SimulatedData.SimulatedDataUsageType` property to **UsePublishedValue**.
- **Pass/Fail Test**—Set the `Step.Result.Evaluations[i].PassFail.SimulatedData.SimulatedDataUsageType` property to **UsePublishedValue**.

The following table outlines how to specify a single value for all sites and how to specify values per site for a test.

Type of Test	Single Value for All Sites	Specific Value Per Site
Numeric Limit	1. Set the `Step.Result.Evaluations[i].NumericLimit.SimulatedData.SimulatedDataUsageType` property to **AllSites**. 2. Specify the value you want to use in the `Step.Result.Evaluations[i].NumericLimit.SimulatedData.AllSites` property.	1. Set the `Step.Result.Evaluations[i].NumericLimit.SimulatedData.SimulatedDataUsageType` property to **PerSite**. 2. Specify the values you want to use in the `Step.Result.Evaluations[i].NumericLimit.SimulatedData.PerSite` property.
Pass/Fail	1. Set the `Step.Result.Evaluations[i].PassFail.SimulatedData.SimulatedDataUsageType` property to **AllSites**. 2. Specify the value you want to use in the `Step.Result.Evaluations[i].PassFail.SimulatedData.AllSites` property.	1. Set the `Result.Evaluations[i].PassFail.SimulatedData.SimulatedDataUsageType` property to **PerSite**. 2. Specify the values you want to use in the `Result.Evaluations[i].PassFail.SimulatedData.PerSite` property.

Data Logs and Reports in Offline Mode

The following table lists how data logs and reports include the state of Offline Mode to indicate that the file includes real or simulated data.

File	Implementation
STDF log file	`NIDTR:<OfflineMode>Disabled</OfflineMode>`

	`NIDTR:<OfflineMode>Enabled</OfflineMode>`
Lot Summary Report	Offline Mode entry in lot description header
Debug Test Results Log	Offline Mode entry in header
Test Program Performance Analyzer summary log file	Offline Mode entry

Note The CSV Test Results Log does not include the state of Offline Mode.

See Also

Offline Mode Workflow

Debugging Test Programs in Offline Mode (TSM)

Note Although simulated instruments behave as closely as possible to real instruments, measurements and data differ in Offline Mode. Additionally, custom code that processes results, test time performance, instrument functionality, and error reporting might differ in Offline Mode and can result in different behavior of the test program.

You can use standard debugging techniques and tools, such as InstrumentStudio, the Digital Pattern Editor, and standard or custom soft front panels to interact with simulated instruments and data in Offline Mode. You can also use LabVIEW and .NET debugging tools to debug applications.

See Also

Offline Mode Workflow

Debugging TestStand Test Programs

Debugging TSM Test Programs

Disabling Offline Mode (TSM)

Select **Semiconductor Module»Disable Offline Mode** or click the **Disable Offline Mode** button on the TSM toolbar to disable Offline Mode and return to the default behavior of TSM. The **Disable Offline Mode** button icon changes to indicate the current state of Offline Mode and the main menu bar no longer displays the **Offline Mode** indicator. The `OfflineMode` property of the NI_SemiconductorModule_StandardStationSettings data type also indicates the current state of Offline Mode.

When you disable Offline Mode, TSM deletes simulated versions of all the instruments defined in the Offline Mode system configuration file.

See Also

Offline Mode Workflow

Offline Mode System Configuration File XML Structure (TSM)

The Offline Mode system configuration XML schema, located at `<Program Files>\National Instruments\Shared\OfflineMode\SystemConfiguration.xsd`, defines the following structure for an Offline Mode system configuration XML file:

Legend	
R	**<Root element>**
E	**<Element>**
A	`Attribute`

R <SystemConfiguration>

- A `schemaVersion`—Specifies the version of the schema file.
- E **<PartNumber>**—Specifies the part number of the system configuration if the system configuration is based on an STS.
  - A `Value`—Specifies the part number of the STS.
- E **<PXIChassis>**—Defines a PXI chassis and the PXI instruments it contains.

- Number—Specifies the number of the PXI chassis.
- Model—Specifies the model of the PXI chassis.
- **<PXI>**—Defines a PXI instrument and all components of the PXI instrument.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation**, for example, HSD_6570_C2_S03, where InstrumentType is an ASCII description of the instrument, ModelNumber is the model number as defined on ni.com, PXIChassisLocation uses a single digit to identify the PXI chassis (Cx), and SlotLocation uses double digits to identify the slot location (Sxx).

  - Name—Specifies the name of the PXI instrument.
  - Model—Specifies the model of the PXI instrument, for example, NI PXIe-6570.
  - Slot—Specifies the slot number of the PXI instrument.
  - NeedsIvi—(Optional) Specifies whether the PXI instrument requires IVI logical names and IVI driver sessions. Set the attribute value to True to create IVI logical names and IVI driver sessions for the PXI instrument.
  - **<PortControlModule>**—Defines a port control module and the port modules the port control module controls. A PXI instrument can contain only one port control module.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation_PCM**, for example, VST_5840_C3_S10_PCM, where InstrumentType is an ASCII

description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), and `SlotLocation` uses double digits to identify the slot location (`Sxx`).

- A `Name`—(Optional) Specifies the name of the port control module. If you do not specify a name, Offline Mode uses the instrument naming convention to provide a name.
- A `Model`—Specifies the model of the port control module, for example, `NI STS-5532`.
- E **<PortModule>**—Defines a port module, cascaded port modules, and port configurations.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation_PCM_PortModule**, for example, `VST_5840_C3_S10_PCM_PM1`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), `SlotLocation` uses double digits to identify the slot location (`Sxx`), and `PortModule` uses a single digit to identify the port module under the port control module (`PMx`).

- A `Name`—(Optional) Specifies the name of the port module. If you do not specify a name, Offline Mode uses the instrument naming convention to provide a name.
- A `Model`—Specifies the model of the port module, for example, `NI STS-5531`.

- A `DigitalSlot`—Specifies the digital slot number of the port module.
- A `AnalogChannel`—Specifies the analog channel of the port module.
- E **<Port>**—(Optional) Defines a port configuration. If you do not specify a port configuration, STS Maintenance Software uses the default port type (for STS-5531) or not connected (for STS-5533 and STS-5534).
  - A `Number`—Specifies the number of the port. Port numbering starts at `0`.
  - A `Type`—Specifies the type of the port.

Model	Number	Supported Types
NI STS-5531	0, 1, 2, 3	`NI5531` (Optional)
NI STS-5533	0, 1, 2, 3	`NI5533_DRA`
NI STS-5534	0, 1, 2, 3	`NI5534_RX` `NI5534_TX`

- E **<PortModule>**—Defines a cascaded port module and port configurations. A port module can contain only one cascaded port module.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation_PCM_PortModule**, for example, `VST_5840_C3_S10_PCM_PM2`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), `SlotLocation` uses double digits to identify the slot location (`Sxx`), and `PortModule`

uses a single digit to identify the port module under the port control module (PMx).

- Name—(Optional) Specifies the name of the port module. If you do not specify a name, Offline Mode uses the instrument naming convention to provide a name.
- Model—Specifies the model of the port module, for example, NI STS-5531.
- DigitalSlot—Specifies the digital slot number of the port module.
- **<Port>**—(Optional) Defines a port configuration. If you do not specify a port configuration, STS Maintenance Software uses the default port type (for STS-5531) or not connected (for STS-5533 and STS-5534).
  - Number—Specifies the number of the port. Port numbering starts at 0.
  - Type—Specifies the type of the port.

Model	Number	Supported Types
NI STS-5531	0, 1, 2, 3	NI5531 (Optional)
NI STS-5533	0, 1, 2, 3	NI5533_DRA
NI STS-5534	0, 1, 2, 3	NI5534_RX NI5534_TX

- **<MmRadioHead>**—Defines a mmWave Radio Head and the mmWave Switches the mmWave Radio Head contains. A PXI instrument can contain up to two mmWave Radio Heads.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation_MMRadioHead**, for example, IF_3622_C2_S04_RH1, where InstrumentType is an ASCII description of the instrument, ModelNumb

er is the model number as defined on ni.com, PXIChassisLocation uses a single digit to identify the PXI chassis (Cx), SlotLocation uses double digits to identify the slot location (Sxx), and MMRadioHead uses a single digit to identify the mmWave Radio Head (RHx).

- Name—(Optional) Specifies the name of the mmWave Radio Head. If you do not specify a name, Offline Mode uses the instrument naming convention to provide a name.
- Model—Specifies the model of the mmWave Radio Head, for example, NI mmRH-5581.
- Number—Specifies the number of the mmWave Radio Head. mmWave Radio Head numbering starts at 0.
- **<MmSwitch>**—Defines a mmWave Switch. A mmWave Radio Head can contain up to two mmWave Switches.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation_MMRadioHead_MMSwitch**, for example, IF_3622_C2_S04_RH1_SW1, where InstrumentType is an ASCII description of the instrument, ModelNumber is the model number as defined on ni.com, PXIChassisLocation uses a single digit to identify the PXI chassis (Cx), SlotLocation uses double digits to identify the slot location (Sxx), MMRadioHead uses a single digit to identify the mmWave Radio Head (RHx), and MMSwitch uses a single digit to identify the mmWave Switch (SWx).

- Name—(Optional) Specifies the name of the mmWave Switch. If you do not specify a name, Offline Mode uses the instrument naming convention to provide a name.
- Model—Specifies the model of the mmWave Switch, for example, NI mmSW-2795.
- Number—Specifies the number of the mmWave Switch. mmWave Switch numbering starts at 0.

- **<Devices>**—Specifies instruments not contained in a PXI chassis.
  - **<USB>**—Defines a USB instrument.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_USB**, for example, DIO_6509_USB, where InstrumentType is an ASCII description of the instrument and ModelNumber is the model number as defined on ni.com.

    - Name—Specifies the name of the USB instrument.
    - Model—Specifies the model of the USB instrument, for example, NI USB-6509.
  - **<PCI>**—Defines a PCI instrument.

Note Consider using the following instrument naming convention for semiconductor test programs: **InstrumentType_ModelNumber_PCI**, for example, MIO_6221_PCI, where InstrumentType is an ASCII description of the instrument and ModelNumber is the model number as defined on ni.com.

    - Name—Specifies the name of the PCI instrument.
    - Model—Specifies the model of the PCI instrument, for example, NI PCI-6221.

- E **<DeviceAssociations>**—Specifies associations between previously defined instruments.
  - E **<DeviceAssociation>**—Defines an association between a parent instrument and a child instrument for a specific purpose.
    - A `ParentDeviceName`—Specifies the name of the parent instrument. The value must match the value of the `Name` attribute of a `<PXI>`, `<USB>`, or `<PCI>` element.
    - A `ChildDeviceName`—Specifies the name of the child instrument. The value must match the value of the `Name` attribute of a `<PXI>`, `<USB>`, or `<PCI>` element.
    - A `Purpose`—Specifies the purpose of the association. The value must be `Digitizer`, `Baseband`, `LO`, `RFConditioner`, `RFInLO`, or `RFOutLO`.

Example

```
<SystemConfiguration>
<PXIChassis Number="3" Model="NI PXIe-1095">
<PXI Name="VST_5820_C3_S02" Model="NI PXIe-5820" Slot="2"
/>
<PXI Name="IF_3622_C3_S04" Model="NI PXIe-3622" Slot="4">
<MmRadioHead Model="NI mmRH-5581" Number="0">
<MmSwitch Model="NI mmSW-2795" Number="0"/>
<MmSwitch Model="NI mmSW-2795" Number="1"/>
</MmRadioHead>
<MmRadioHead Model="NI mmRH-5581" Number="1">
<MmSwitch Model="NI mmSW-2795" Number="0"/>
</MmRadioHead>
</PXI>
```

```
<PXI Name="LO_5653_C3_S06" Model="NI PXIe-5653" Slot="6" />
</PXIChassis>
<DeviceAssociations>
<DeviceAssociation ParentDeviceName="IF_3622_C3_S04" ChildDeviceName="VST_5820_C3_S02" Purpose="Baseband" />
<DeviceAssociation ParentDeviceName="IF_3622_C3_S04" ChildDeviceName="LO_5653_C3_S06" Purpose="LO" />
</DeviceAssociations>
</SystemConfiguration>
```

Schema Version Policy (TSM)

The schema version uses a **major.minor** notation. The version of the schema reflects changes to the schema, not changes to TSM.

Changes to the schema version indicate breaking changes to the schema. TSM does not load files that use a later schema version than the version specified for that version of TSM.

Use the following table to map schema versions and TSM versions.

Schema Version	TSM Version
PinMap.xsd version 1.0 BinDefinitions.xsd version 1.0	NI TestStand 2013 Semiconductor Module
PinMap.xsd version 1.1 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module
PinMap.xsd version 1.2 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2014 Semiconductor Module SP1 NI TestStand 2016 Semiconductor Module
PinMap.xsd version 1.3 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2016 SP1 Semiconductor Module

PinMap.xsd version 1.4 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0	NI TestStand 2017 Semiconductor Module
PinMap.xsd version 1.5 BinDefinitions.xsd version 1.1 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.0	NI TestStand 2019 Semiconductor Module
PinMap.xsd version 1.6 BinDefinitions.xsd version 1.2 Specifications.xsd version 1.0 SystemConfiguration.xsd version 1.1	NI TestStand 2020 Semiconductor Module TestStand 2020 Semiconductor Module 2021 Q4

TSM Sequence Analyzer Rules Descriptions

TSM includes the following general, performance, best practices, and statistics rules to use in the TestStand Sequence Analyzer in the TestStand Sequence Editor or the stand-alone sequence analyzer application. The built-in TSM rules are enabled by default. You can also create and deploy custom rules and analysis modules.

The following sections describe the built-in TSM rules, listed in order of severity.

- General
- Performance
- Best Practices
- Statistics
- Semiconductor Sequence Call Step

General

Rule	Severity	Description
No DUT pins have been specified	Error	The **Specify DUT Pins** checkbox is enabled, but no DUT pin has been specified in the **Specify DUT Pins** section on the Options tab of the Step Settings pane of a Semiconductor Multi Test step.

Step specifies invalid DUT pins	Error	The **Specify DUT Pins** option on the Options tab of a Semiconductor Multi Test step or a Semiconductor Action step is enabled, but some of the included DUT pins are not in the pin map file.
Single limit missing	Error	A limit is missing on a Numeric Limit test in a Semiconductor Multi Test step.
PAT (Part Average Testing) settings invalid	Error	One or more PAT algorithm settings for the test program are invalid. Correct the settings on the PAT Algorithm Settings panel of the Test Program Editor.
Pin name is invalid	Error	A pin specified by the test in a Semiconductor Multi Test is not present in the pin map the test program specifies.
Test step properties are not assignable	Error	A step in the test program is setting the values of properties (such as TestNumber, FailBin, and so on) of a Semiconductor Multi Test step. Setting a property on a Semiconductor Multi Test step at run time does not change its behavior because the step does not access step properties after the first time it executes.
Instruments defined in pin map are missing from MAX	Error	Some of the instruments defined in the pin map are missing from Measurement & Automation Explorer (MAX), which can prevent the test program from running. Correct instrument names in the pin map or add the correct instruments in MAX.
Alarms settings are invalid	Error	The alarms settings do not match the pin configuration specified in the pin map. This can occ

		ur when the pin map is modifie d after the alarms settings have been configured. It can also occ ur if the configured alarms for t his sequence file are not suppor ted in the current system. Use t he Alarms panel of the Test Pro gram Editor to update the alar ms settings.
Test number missing	Warning	A test in a Semiconductor Multi Test step does not specify a test number.
Pin map invalid	Warning	The specified pin map is invalid or the pin map has not been sp ecified. Use the Pin Map panel o f the Test Program Editor to spe cify the pin map file to use in th e test program.
Software bin missing	Warning	A test in a Semiconductor Multi Test step does not specify a soft ware bin.
Bin definitions invalid	Warning	The specified bin definitions is i nvalid or the bin definitions has not been specified. Use the Bin Definitions panel of the Test Pro gram Editor to specify the bin d efinitions file to use in the test p rogram.
Sessions created with LabVIEW steps used in .NET steps or vice-versa	Warning	A Semiconductor Multi Test or S emiconductor Action .NET step uses an instrument driver sessi on that a Semiconductor Multi Test or Semiconductor Action L abVIEW step created or vice-ver sa.
Both limits missing	Information	A test in a Semiconductor Multi Test step does not specify any li mits. For tests that have no limi ts, the Semiconductor Multi Tes

		t step only logs the data and do es not pass or fail the test.

Performance

Rule	Severity	Description
LabVIEW Adapter in Developme nt System mode	Warning	Configuring the LabVIEW Adapt er to use the **LabVIEW Develo pment System** can negatively affect performance. Configure t he LabVIEW Adapter to use the **LabVIEW Runtime** if the LabV IEW Development System mod e is not required.
Minimal use of system pins	Warning	A Semiconductor Multi Test ste p includes system pins. Includin g system pins forces the step co de module to execute in a singl e thread and can negatively affe ct performance. Ignore this war ning for steps that need to acce ss system pins.
Non-reentrant code module	Warning	A Semiconductor Multi Test ste p or Semiconductor Action step calls a VI with the **Reentrancy** option on the Execution page of the VI Properties dialog box set to **Non-reentrant execution** or **Preallocated clone re-ent rant execution**. Using these re -entrancy options can negativel y affect performance because L abVIEW might clone the VI each time it executes.
Disable result recording option	Information	Enabling the **Result Recordin g Option** for steps that do not i nclude defined tests can negati vely affect performance. Enable result recording while you meas ure performance but disable it t

		o attain optimal performance. This option is located on the Run Options panel of the Properties tab of the Step Settings pane of a step.

Best Practices

Rule	Severity	Description
Avoid overriding PreUUT, PreMainSequence, PostMainSequence callbacks	Error	If a run-time error occurs in one of these callbacks, the end-of-test (EOT) signal is not sent to the handler driver.

Statistics

Rule	Severity	Description
Count tests	Information	The total number of tests in all the analyzed files, including count per sequence, count per sequence file, and count per analysis.

Semiconductor Sequence Call Step

Rule	Severity	Description
Published data IDs should not be duplicated on tests in the called sequence	Warning	The 'Published Data Id' field on tests on Semiconductor Multi Test steps in sequences called by Semiconductor Sequence Call steps should not be duplicated. The duplicate tests will be omitted at run time.
Published data IDs should match a test in the called sequence	Warning	Simple IDs specified in the 'Step Name.Published Data Id' field on Semiconductor Sequence Call tests should match the 'Published Data Id' field of a test on a Semiconductor Multi Test step in the called sequence. Specify a value for the 'Step Name.Publis

		hed Data Id' field that matches the 'Published Data Id' field of a test on a Semiconductor Multi Test step in the called sequence.
'Step Name.Published Data Id' should not be empty	Warning	The 'Step Name.Published Data Id' field on Semiconductor Sequence Call tests should not be empty. Specify a value for the 'Step Name.Published Data Id' field that matches the 'Published Data Id' field of a test on a Semiconductor Multi Test step in the called sequence.
'Step Name.Published Data Id' must match a test on the specified step in the called sequence	Error	Values that include step names for the 'Step Name.Published Data Id' field on Semiconductor Sequence Call tests must match the 'Published Data Id' field of a test on the specified Semiconductor Multi Test step in the called sequence. Specify a value for the 'Step Name.Published Data Id' field that matches the 'Published Data Id' field of a test on a Semiconductor Multi Test step in the called sequence.
Step Name.Published Data Id must not match multiple tests in the called sequence	Error	The 'Step Name.Published Data Id' field on Semiconductor Sequence Call tests must not match the 'Published Data Id' field of more than one test on Semiconductor Multi Test steps in the called sequence. Rename one of the steps in the called sequence or include the step name in the 'Step Name.Published Data Id' column to match a single test.

Test Time Reduction and Test System Performance Improvements (TSM)

Before you make test time reduction (TTR) improvements, accurately measure the performance of the test program to evaluate how the changes you make affect the performance of the test program. Select **Semiconductor Module»Measure Performance of <filename>** to run the test program in an operator interface with one or more site configurations and collect performance data. To obtain the most accurate results, execute the test program in the LabVIEW Run-Time Engine (RTE). Use the Test Program Performance Analyzer to view the performance data TSM generates.

Note Executing a test program in a TSM operator interface is typically faster than executing the test program in the TestStand Sequence Editor development environment.

Use the following test time reduction techniques to improve the performance of any test program.

- Disable unnecessary result processors
- Use reentrant VIs
- Execute VIs with the LabVIEW Run-Time Engine
- Use only needed pins
- Enable parallel For Loop iterations in VIs
- Reduce settling times in test code modules
- Disable tracing in TestStand
- Use inline expansion for sequence call steps to reduce overhead of sequence calls

Advanced Techniques

Use the following advanced techniques to further improve performance.

Note Some of these techniques might have a negative impact on performance for some test

programs. Measure the test program performance before and after each change to determine how the change affects performance.

- Use the Execution Profiler to identify which shared tester resources cause throughput bottlenecks
- Perform analysis in parallel with measurements
- Adjust thread settings to maximize parallelism
- Adjust computer BIOS CPU settings

Disable Unnecessary Result Processors (TSM)

For each result processor enabled in the Result Processing dialog box, TestStand calls sequences associated with the result processor to process test results during testing, which can add time to test execution. Although the TestStand Semiconductor Module (TSM) STDF Log generator, Lot Summary Report generator, and CSV Test Results Log generator do not contribute significantly to test execution time, the TestStand Report generator and TSM Debug Test Results Log generator can cause performance and memory usage issues. To limit the effect of the Debug Test Results Log generator, enable the **Limit Number of Results Displayed in Report View** option or the **Log Results Only for DUT Failures** option in the Debug Test Results Log Options dialog box. Disable any result processors you do not need during production testing.

Complete the following steps to disable result processors from the TestStand Sequence Editor.

1. Select **Configure»Result Processing** to launch the Result Processing dialog box.
2. Remove the checkmark from the **Enabled** option for the result processor you want to disable.

Complete the following steps to disable result processors from the TSM operator interface.

1. Click the **Configure Station** button to launch the Configure Station Settings dialog box.

2. On the Advanced tab, click the **Result Processing** button to launch the Result Processing dialog box.
3. Remove the checkmark from the **Enabled** option for the result processor you want to disable.

See Also

TSM Reports and Data Logs

TestStand Result Collection

Use Reentrant VIs (TSM)

When you perform multisite testing, you can improve performance by simultaneously testing multiple sites using multiple execution threads. When you use the TestStand Batch process model to test multiple sites, TestStand executes the test program in separate threads automatically. LabVIEW, however, by default does not allow multiple instances of a VI to execute simultaneously. Each call to a VI must wait until other calls to the VI finish executing. To maximize parallel execution of test code, use reentrant VIs and subVIs.

Complete the following steps to configure a VI to set the execution properties for a VI.

1. From the front panel or block diagram of a VI, select **File»VI Properties** and select **Execution** from the **Category** drop-down menu to launch the Execution page.
2. Set the Reentrancy option to **Shared clone reentrant execution**. Do not use the **Preallocated clone reentrant execution** option because that option might cause LabVIEW to allocate significantly more resources to execute VIs and can result in high memory usage and long delays when unloading the test program.

Refer to the **LabVIEW Help** for more information about reentrant VIs. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

Execute Test Code Modules Using the LabVIEW Run-Time Engine (TSM)

The TestStand LabVIEW Adapter can execute VI code modules in the LabVIEW Development System process or in the TestStand process using the LabVIEW Run-Time Engine (RTE). Executing test code in the LabVIEW Development System process is useful for debugging but adds significant performance overhead to the execution of test programs. To reduce this overhead, execute VIs in the TestStand process using the LabVIEW RTE.

Complete the following steps to configure the LabVIEW Adapter to use the LabVIEW RTE from the TestStand Sequence Editor.

1. Select **Configure»Adapters**.
2. Select the LabVIEW Adapter in the list control and click **Configure** to launch the LabVIEW Adapter Configuration dialog box.
3. Enable the **LabVIEW Run-Time Engine** option.

Complete the following steps to configure the LabVIEW Adapter to use the LabVIEW RTE from the TestStand Semiconductor Module operator interface.

1. Click the **Configure Station** button to launch the Configure Station Settings dialog box.
2. On the Advanced tab, click the **LabVIEW Adapter** button to launch the LabVIEW Adapter Configuration dialog box.
3. Enable the **LabVIEW Run-Time Engine** option.

Configure Semiconductor Steps to Use Only Needed Pins

Typically, you use the TestStand Batch process model to perform multisite testing in parallel. However, when you connect a DUT pin to an instrument shared by multiple sites, and the instrument does not permit independent operations on its channels, the Semiconductor Multi Test or Semiconductor Action step executes in one test socket to test multiple sites. In this situation, the step determines if it can execute tests in parallel by checking that the pins the test code module uses are connected to instruments that are not shared across multiple sites.

By default, the Semiconductor Multi Test and Semiconductor Action steps assume a code module uses every DUT pin. To improve performance, specify the DUT pins that the test code module uses.

1. In the sequence editor, click the step you want to edit.
2. Click on the Options tab in the Step Settings pane.
3. Select **Specify Manually** in the **Specify Pins and Relays** drop-down menu.
4. To specify the DUT pins, enable the **Specify DUT Pins** option.
5. Enable the DUT pins and pin groups the step uses.
6. Review the **Multisite Execution Diagram** to determine how many threads TSM uses to execute the step.

If the test uses any system pins, enable the **Include System Pins** option. When you enable the **Include System Pins** option, the step executes in one thread for all sites.

Note Specifying DUT and system pins on steps in the `ProcessSetup` and `ProcessCleanup` sequences has no effect. Steps in those sequences execute as if you selected **One thread only** in the **Multisite Option** drop-down menu and included all DUT and system pins in the `SemiconductorModuleContext` object.

See Also

Multisite Programming Techniques

Subsystems and Pin Maps

Enable Parallel For Loop Iterations in VIs (TSM)

In many cases, VIs that test multiple sites or pins contain For Loop structures that perform computationally expensive or time consuming operations for multiple instruments. Often, you can improve execution speed by enabling parallel iterations

on the For Loop structure. Not all For Loops can run with parallel iterations, and in some cases, a For Loop does not benefit from parallelization. Additionally, you can adjust thread settings for parallel For Loops to improve performance.

Refer to the **Parallel Iterations: Improving For Loop Execution Speed** topic in the **LabVIEW Help** to determine whether you can improve performance by enabling parallel For Loop iterations in VI test code modules. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

See Also

Thread Settings for Maximum Parallelism

Parallel For Loops

Reduce Settling Times in Test Code Modules (TSM)

It is common to add a settling time before taking measurements to achieve the required precision. Once a test program is running well, evaluate the settling times to reduce them to the minimum settling time required for an accurate measurement.

Disable Tracing in TestStand (TSM)

Enabling tracing to display each step as it executes is useful for debugging but adds significant performance overhead to the execution of test programs in the TestStand Sequence Editor and in the TestStand Semiconductor Module (TSM) operator interface. TSM automatically disables tracing for operator interfaces. Typically, you do not need to disable tracing unless a custom operator interface enables tracing.

Complete the following steps to disable tracing from the sequence editor.

1. Select **Configure»Station Options** to launch the Station Options dialog box.
2. On the Execution tab, disable the **Enable Tracing** option.

Use Inline Expansion for Sequence Call Steps

TestStand Sequence Call and Semiconductor Sequence Call steps allow you to simplify editing and maintaining test programs by placing common test steps into sequences. However, these Sequence Call steps incur a performance penalty that affects the overall test time. You can eliminate this performance penalty at the expense of a more complex test program by using inline expansion. Inline expansion is the process of replacing Sequence Call steps with the contents of the sequences they call.

Follow these steps to use inline expansion with a TestStand or Semiconductor Sequence Call step in the `MainSequence` sequence:

1. Replace the Sequence Call step in the `MainSequence` sequence with the contents of the sequence that the Sequence Call step calls.
2. Rename the steps you inserted in the `MainSequence` sequence, if necessary, to avoid duplicate step names.
3. For each parameter in the called sequence, replace all references to the parameter in the step expressions of the steps you copied with the value that the Sequence Call step used for the parameter.
   For example, replace the expression `Parameters.Frequency` with **3000** if the Sequence Call step passes 3000 for the `Frequency` parameter.
4. Copy local variables from the called sequence into the `MainSequence` sequence.
5. Rename the local variables you copied, if necessary, to avoid duplicate variable names. Update references to the local variables with the new names.
6. Complete the following actions for each Semiconductor Action and Semiconductor Multi Test step you copied to the `MainSequence` sequence:
   a. Click on the Options tab for the step.
   b. Check if the step uses an expression to specify the pins.
   c. If the step uses an expression to specify the pins, choose **Select Manually** from the **Specify Pins and Relays** drop-down menu and specify the pins explicitly using the check boxes.

7. Complete the following actions for each Semiconductor Multi Test step you copied to the `MainSequence` sequence:
    a. Click on the Tests tab for the Semiconductor Multi Test step.
    b. Identify tests that require additional configuration and note their published data IDs.
    c. Click on the Tests tab of the Semiconductor Sequence Call step.
    d. Find the tests with the same published data IDs you noted in step 7b.
    e. Copy the tests from the Semiconductor Sequence Call to the Semiconductor Multi Test step.

Thread Settings for Maximum Parallelism (TSM)

Depending on the number of sites a test program tests, the number of pins in the pin map, the number of independent instrument resources on the tester, and the number of logical processors on the tester, you might need to adjust several thread-related settings to achieve greater execution parallelism and better performance. Because using the suggested setting values does not guarantee optimal performance, measure test program performance each time you change these settings to determine how the change affects performance.

LabVIEW Parallel For Loops

LabVIEW creates multiple instances of the contents of parallel For Loops and executes them in parallel. The number of instances that execute in parallel is determined by the value wired to the **P** terminal and the value of the **Number of generated parallel loop instances** option in the For Loop Iteration Parallelism dialog box in LabVIEW. By default, LabVIEW uses the minimum of the number of logical processors on the tester and the number of logical processors on the computer used to save the VI. To ensure maximum parallelism, use the following parallel For Loop settings:

- **Number of generated parallel loop instances**—The best value for this setting depends on the purpose of the loop. Do not use an arbitrarily large number for this setting because it can add to execution time and can cause high memory usage and long delays when unloading the test program.

- For the For Loops that iterate over instrument sessions, set the **Number of generated parallel loop instances** option to the maximum number of instrument sessions the loop can use. The maximum number of instrument sessions used depends on the type of instrument, the number of pins, and the number of sites and settings on the Options tab of the Semiconductor Multi Test or Semiconductor Action step. For channel-based sessions, such as NI-DCPower sessions, use the following equation to determine the maximum number of sessions the loop can use:

```
(number of pins used by loop)*(maximum number of sit
es in a single block in the Multisite Execution Diag
ram on Options tab)
```

For NI-Digital Pattern instruments and other instrument-based sessions, use the number of instruments of that type in the pin map.

- For the For Loops that process data, you often can achieve the best performance by disabling loop parallelism. If the data processing is relatively slow and CPU intensive (such as the computation of large FFTs), first set the **Number of generated parallel loop instances** option to the number of logical processors on the target tester. If the loop has more iterations than the number of logical processors, try larger values for the **Number of generated parallel loop instances** option and measure performance to determine the best value to use.

- **P terminal**—Wire the **P** terminal to the size of the array that the For Loop uses for indexing. Leaving the **P** terminal unconnected causes LabVIEW to use the lesser of the **Number of generated parallel loop instances** option and the number of logical processors on the tester.

LabVIEW Adapter Settings

For VIs that TestStand steps call using the LabVIEW Run-Time Engine, the LabVIEW Adapter allocates threads to execute the VIs that use the **same as caller** execution system. Change the **Number of Threads** option on the LabVIEW Adapter Configuration dialog box to the maximum number of parallel loop instances that all VIs in the test program require.

Note In TestStand 2016 and earlier, you can use the LabVIEW Adapter Configuration dialog box to set the number of LabVIEW execution threads to a maximum value of 2 x the number of logical processors. If you need more threads than that for all VIs running in parallel, configure the VIs to use an execution system other than the **same as caller** execution system on the Execution page of the VI Properties dialog box in LabVIEW and configure the number of execution threads assigned to a LabVIEW execution system.

The following versions of TestStand prevent you from setting the **Number of Threads** option in the LabVIEW Adapter Configuration dialog box greater than a maximum value:

Version	Maximum Value
TestStand 2016 SP1 and earlier	12
TestStand 2017	36
TestStand 2017 SP1 and later	96

If you need more threads than the maximum number permitted, you must edit the `<TestStand Application Data>\Cfg\Adapters.cfg` file and change the value of the `_FlexGAdp_NumofThreadsUsedinRTE` setting. A limitation in LabVIEW 2018 SP1 and earlier might prevent you from setting a large number of threads.

Adjust BIOS CPU Settings (TSM)

Some configuration options can lead to slower than expected test times when using the TestStand Semiconductor Module. You might be able to improve performance of a test program by disabling hyper-threading or by reducing the number of cores the

CPU uses. Always measure performance before and after making any changes to determine how the change affects performance.

Use the BIOS settings on the computer to disable hyper-threading or to change the number of CPU cores. Typically, you enter the BIOS settings screen by pressing the <Delete> (DEL) key after you restart the computer.

Measuring Performance (TSM)

You can use TSM, LabVIEW, and TestStand tools to measure execution times so you can determine how changes you make affect performance.

TSM

During the test program development phase, you can use built-in TSM tools to measure test program performance and then analyze the resulting data with the Test Program Performance Analyzer. Some common use cases include identifying the slowest test times, identifying low parallel test efficiency (PTE) values, and displaying the overall socket time and the calculated PTE value for each site configuration.

One way to measure overall performance of the test system is to test a number of DUTs using the TSM operator interface and study the values of the Socket Time and Cycle Time controls. The socket time primarily corresponds to the time it takes to perform test code. The cycle time includes the socket time and the time required to perform other tasks, such as binning, placing DUTs, and generating reports.

When you measure performance, test enough DUTs to obtain consistent results and minimize the number of processes running simultaneously that could affect the performance of the TSM operator interface. The first batch in a lot typically takes longer to load than the following batches because you must also load resources.

Note Executing a test program in a TSM operator interface is typically faster than executing the test program in the TestStand Sequence Editor development environment.

LabVIEW

Use the following tools to help analyze the performance of VIs:

- **LabVIEW VI Analyzer**—A static code analyzer that identifies potential performance problems.
- **LabVIEW Desktop Execution Trace Toolkit**—Performs dynamic code analysis to identify problems that might negatively affect performance, such as memory leaks and reference leaks.
- **Profile Performance and Memory window**—Built-in LabVIEW tool that determines how an individual VI spends time and how the VI uses memory.

Complete the following steps to use the Profile Performance and Memory window:

1. Configure the TestStand LabVIEW Adapter to execute VIs in the LabVIEW Development System process.
2. Open the LabVIEW project that contains the VIs you want to analyze.
3. Select **Tools»Profile»Performance and Memory** to launch the Profile Performance and Memory window.
4. Click the **Start** button to begin collecting performance data.
5. Use the TSM operator interface to test a number of DUTs to execute VIs you want to profile multiple times.
6. Click the **Snapshot** button in the Profile Performance and Memory window to acquire execution time data for the VIs.

TestStand

During test program development, use the Execution Profiler to view and record duration the of steps, code modules, and other resources a multithreaded TestStand system uses over a period of time. For example, you can identify parts of the test program that take longest to execute or identify what shared tester resources cause throughput bottlenecks.

You can also use the Model Plug-in - Basic Step Time Report example tool to measure step execution times. The example includes a sample report generator that generates a report in Microsoft Excel format that includes time data for individual steps. Use the TestStand Execution Profiler to measure execution times of steps and code modules.

Test Program Performance Analyzer (TSM)

Use the Test Program Performance Analyzer to view data TSM generates when you measure the performance of a test program. You can filter, graph, compare, and save the data in various ways to identify performance issues in a test program.

The Test Program Performance Analyzer automatically launches after you complete a test program performance measurement and loads the data TSM generates. You can also select **Semiconductor Module»Launch Test Program Performance Analyzer** to open a new instance of the Test Program Performance Analyzer without preloaded data.

Mode

The Test Program Performance Analyzer graph displays data in the following modes:

- **Single Data Set** (default)—Analyzes a single data log file. A single data log file can contain data from multiple test program performance measurement operations using different multisite configurations. A test program performance measurement can contain a single execution on a particular site configuration or it can contain multiple executions with varying site configurations.
- **Compare Data Sets**—Analyzes two log files to compare performance. Select this option when you want to analyze how modifications to a test program affect performance.

The **Plot Legend** in the upper right corner of the analyzer displays a description of each plot the graph displays.

See Also

Common Use Cases for Measuring Test Program Performance

Measuring Test Program Performance (TSM)

During the test program development phase, you can run a series of executions with varying site configurations to characterize the performance of a test program as the site configuration changes.

Complete the following steps to measure test program performance using a consistent environment.

1. Open the test program you want to measure. The test program must use the Batch or Sequential process model.
2. Verify that the test program specifies a pin map.
3. Select **Semiconductor Module»Measure Performance of <filename>**.
4. Set the options in the Test Program Performance Measurement Configuration dialog box that launches.
5. Click the **Configure Lot Settings** button to launch the Configure Lot Settings dialog box. Configure the settings and click **OK**.

   Note The Configure Lot Settings dialog box does not display the Enabled Sites list because you use the Test Program Performance Measurement Configuration dialog box to specify site configurations.
6. Click **OK** in the Test Program Performance Measurement Configuration dialog box.
7. The test program executes in the operator interface that launches. TSM logs the test program performance data during execution. TSM automatically discards performance data from the first batch in a lot.
8. When the test program execution completes, the operator interface automatically closes, and TSM launches the Test Program Performance Analyzer for you to view and analyze the resulting data.

See Also

Common Use Cases for Measuring Test Program Performance

Common Use Cases for Measuring Test Program Performance (TSM)

Use the Test Program Performance Analyzer to complete the following common tasks.

Identify Slow Test Times

1. Load a data set log file.
2. Select **Test Time** in the **Sorting Method** section of the analyzer.
3. The analyzer sorts the steps by the highest average test time of the maximum number of sites configuration.
4. Select **Code Module Time** in the **Timing Method** section of the analyzer to view times for changes you make to code modules in the test program.
5. Click a bar in the graph to display a graph of the single step times for that step name and the number of sites, which can be helpful to determine if unexpected poor performance is related to particular cycles or sites.

Identify Low Parallel Test Efficiency (PTE) Values

1. Load a data set log file.
2. Select **PTE** in the **Sorting Method** section of the analyzer.
3. The analyzer sorts the steps by the worst calculated PTE values.
4. The bars on the graph display the test time, and the diamonds on the graph display the calculated PTE of the test time relative to the site configuration test time.

Display Socket Time and Calculated PTE Values

1. Load a data set log file.
2. Select **Socket Times/PTE** in the **Graph Type** control of the analyzer.

3. The analyzer displays the overall socket time and the calculated PTE value for each site configuration.
4. Use the **Target Test Time** section of the analyzer to enter the target PTE% you want to reach and to specify the single site test time to use to calculate the target test time.
5. The graph displays the target times that each site configuration would need to meet to achieve the PTE% you set.

Test Program Performance Measurement Testing Environment (TSM)

To ensure that the testing environment is consistent while you measure test program performance, the `NI_SemiconductorModule_PerformanceMeasurement.seq` file configures the environment in the following ways when TSM launches the operator interface you specify to simulate a production environment. The sequence file is located in the `<TestStand>\Components\Modules\NI_SemiconductorModule` directory.

- Configures the LabVIEW Adapter to use the LabVIEW Run-Time Engine. After testing completes, restores the original setting for the adapter.
- Enables or disables result processors. After testing completes, restores the original settings for the result processors.
  - Enables the following result processors:
    - `NI_SemiconductorModule_LotSummaryReportGenerator.seq`
    - `NI_SemiconductorModule_StdfGenerator.seq`
  - Disables the following result processors:
    - `NI_DatabaseLogger.seq`
    - `NI_OfflineResultsGenerator.seq`
    - `NI_ReportGenerator.seq`
    - `NI_SemiconductorModule_LotTestingComplete.seq`

    - `NI_SemiconductorModule_TestResultsLogGenerator.seq`
  - The Test Program Performance Analyzer uses its own simulated handler driver and does not support the NI Built-in Simulated Handler Driver or any custom handler drivers.

Modifying the Testing Environment

Complete the following steps to modify the testing environment to simulate a specific production environment.

- Copy the `NI_SemiconductorModule_ PerformanceMeasurement.seq` file from the `<TestStand>\Components\Modules\NI_SemiconductorModule` directory and make changes to the copy.
- Modify the steps in the Setup and Cleanup step groups to meet the needs of the environment you want to simulate. For example, the Configure Result Processors step contains arrays of result processor names and enabled states. You can modify the step to include additional result processors or remove existing ones.
- Save the changes to the sequence file.
- In the Test Program Performance Measurement Configuration dialog box, modify the **Operator Interface Command Line** option in the **Advanced** section by replacing the path to the `NI_SemiconductorModule_PerformanceMeasurement.seq` file with the path to the new sequence file you just created and modified.

Test Program Performance Measurement Configuration Dialog Box (TSM)

Select **Semiconductor Module»Measure Performance of <filename>** in the TestStand Sequence Editor to launch the Test Program Performance Measurement Configuration dialog box, in which you can specify settings for measuring test program performance. The changes you make persist in the dialog box.

The Test Program Performance Measurement Configuration dialog box contains the following options:

- **Test Program**—Displays the test program on which to measure performance.
- **Process Model**—Displays the process model the test program uses.
- **Operator Interface Path**—By default, specifies the absolute path to the default LabVIEW operator interface TSM installs. Use the browse button to navigate to a custom operator interface to use instead of the default TSM LabVIEW operator interface or paste the path to the executable into the control. This control cannot be empty.
- **Output File Directory**—Specifies the directory in which to write the log files. Use the browse button to navigate to a custom output directory or paste the path to the directory into the control. The path must be an absolute path.
- **Number of Parts Per Site**—Specifies the number of parts to run on each site.
- **Site Configurations**—Each row corresponds to a lot and contains a comma-separated list of integers or integer ranges that specifies the sites the lot contains, for example: 0, 1, 2-7. The sites must exist in the pin map. Use the **Add Site Configuration** and **Delete Site Configuration** buttons to add and delete rows. TSM ignores empty rows at run time.

Note If you are using the Sequential process model, **Site Configurations** displays a single configuration that contains a single site (0). If you switch back to using the Batch process model, **Site Configurations** displays the previous set of configurations.

- **Configure Lot Settings**—Launches the Configure Lot Settings dialog box. If you have not configured lot settings for the active test program, TSM disables the OK button and displays a warning to prompt you to configure the lot settings.

Click the expand/collapse button to view or hide the **Advanced** section of the dialog box, which includes the following option that configures the testing environment to simulate a production environment:

- **Operator Interface Command Line**—The default command line is `/NoCompatibilityIssuesDialog /run "Measure Test Program Performance" "<TestStand>\Components\Modules\NI_SemiconductorModule\NI_SemiconductorModule_PerformanceMeasurement.seq" /quit`, which configures the execution environment. If you use a custom sequence file instead of the default `NI_SemiconductorModule_PerformanceMeasurement.seq` file to configure the environment for performance testing, modify the command line to call the custom sequence file.

Note If you use an operator interface with a custom command-line parser, modify the command-line arguments to ensure that the operator interface calls the `Measure Test Program Performance` sequence of the `NI_SemiconductorModule_ PerformanceMeasurement.seq` file located in the `<TestStand>\Components\Modules\NI_SemiconductorModule` directory.

- **Reset to Default**—Reverts the value of the **Operator Interface Command Line** option to use the default command-line value.
- **Allow execution with LabVIEW Development Environment Adapter**—By default, when you select the LabVIEW Development Environment adapter in TestStand, the adapter temporarily switches to use the LabVIEW Run-Time Engine to simulate production performance. Enabling this option disables the switch to the LabVIEW Run-Time Engine, allowing performance measurement with the LabVIEW Development Environment adapter. This increases test time. This option is disabled if the LabVIEW adapter is already set to use the LabVIEW Run-Time Engine.

Note Changing any option under **Advanced** causes the section to remain

expanded until the next time the dialog box is opened.

Loading Data in the Test Program Performance Analyzer (TSM)

The Test Program Performance Analyzer automatically launches after you complete a test program performance measurement and loads the data TSM generates.

If an instance of the Test Program Performance Analyzer is already running when you complete a test program performance measurement, the most recently launched instance of the analyzer handles the data TSM generates in the following ways:

- If the analyzer is not displaying any data, displays the newly generated data set.
- If the analyzer is displaying a single data set, prompts you to compare the newly generated data set to the previous data set displayed in the analyzer or to replace the previous data set displayed in the analyzer with the newly generated data set.
- If the analyzer is already comparing two data sets, prompts you to compare the newly generated data set to the previous base data set displayed in the analyzer or to the previous modified data set displayed in the analyzer.
  - When you click the **Compare with Base Data** button, TSM replaces the previous modified data set displayed in the analyzer with the newly generated data set.
  - When you click the **Compare with Modified Data** button, TSM completes the following actions:
    - Discards the previous base data set displayed in the analyzer.
    - Converts the previous modified data set displayed in the analyzer to the current base data set to display in the analyzer.
    - Replaces the previous modified data set displayed in the analyzer with the newly generated data set.

You can also use the **Log File Path** control and **Load** button near the upper right corner of the Test Program Performance Analyzer to manually specify and load a log file for a single data set to display in the analyzer. When you select the **Compare Data Sets** mode, you can use the **Base Log File Path** and the **Modified Log File Path** controls and **Load** buttons to manually specify and load log files for two data sets to display and compare in the analyzer.

Loading large log files can cause performance and memory usage issues. To improve performance and avoid memory usage issues, load a sample of the batches in a log file by enabling the **Log Data Sample Rate** checkbox and specifying the sample rate. When the **Log Data Sample Rate** checkbox is enabled, one out of every N batches in a log file will be loaded when you click the **Load** button, where N is the specified sample rate.

To view the metadata of log files, including notes, click the **Log Browser** button to open the Log Browser Window.

See Also

Log Browser Window

Test Program Performance Analyzer Graphs (TSM)

Use the Test Program Performance Analyzer to visualize performance log data. Graph types include step times statistics, single step times, scatter distribution, and parallel test efficiency (PTE). Use the **Graph Type** control to select the type of graph to display. Use the **Timing Method** section of the analyzer to specify timing options for the graph.

Step Times Statistics Graph

Select **Step Times Statistics** in the **Graph Type** control to display a graph you can use to analyze performance bottlenecks by visualizing statistical information on

a step's test times. Use the **Sorting Method** section of the analyzer to specify sorting options for the graph. By default, TSM enables the following options:

- **Show Average Step Times**—Displays a colored bar indicating the average time it takes for a step to run. To hide this plot, remove the checkmark from the **Show Average Step Times** checkbox.
- **Show PTE%**—The top x-axis of the graph displays the PTE percentage of the step. The bottom x-axis of the graph displays the step time in milliseconds. A diamond point over the bars in the graph displays the calculated PTE for each step and number of sites. To hide this plot, remove the checkmark from the **Show PTE%** checkbox.
- **Show Distribution Box Plot**—Displays the statistical distribution of step times as a box plot. To hide this plot, remove the checkmark from the **Show Distribution Box Plot** checkbox.

Hover over a bar or box plot in the graph to display a tooltip with the following information:

- Step name
- Number of Sites (in the multisite configuration)
- Average (if **Show Average Step Times** is enabled)

When you select **Compare Data Sets** in the **Mode** section of the analyzer, the value in the tooltip includes the value of the bar the mouse is hovering over, the matching value in the other log file, and the difference between the two values.

The **Show Distribution Box Plot** option displays the following additional statistical distribution information at the bottom of the tooltip:

- Minimum
- Lower Quartile
- Median
- Upper Quartile
- Maximum

Click a bar in the graph to display a graph of the Single Step Times for that step name and the number of sites. To return to the Step Times Statistics graph, click the back button next to the title in the upper left corner of the graph.

Click a plot name to the left of the graph area to display a scatter graph of the step times for that step name. To return to the Step Times Statistics graph, click the back button next to the title in the upper left corner of the graph.

When you right-click the step name, a context menu opens. Click **Go to Step in Sequence Editor** to navigate to the step's location in the sequence editor. If the sequence file cannot be found, or if the step or sequence have been deleted, the sequence editor displays an error message.

Single Step Times Graph

Select **Single Step Times** in the **Graph Type** control to display a graph you can use to view the time it takes for one step to run. Use the corresponding options in the **Graph Type** section of the analyzer to specify the step name and number of sites to display in the graph for the step. You can also click a step in the Step Times Statistics graph to display the graph for the step.

The upper x-axis of the graph displays the site. It also displays the data set log data file when you select **Compare Data Sets** in the **Mode** section of the analyzer, using the letter B to indicate the base data set log file and M to indicate the modified data set log file. The lower x-axis of the graph displays the batch index number.

The graph groups sites by subsystem and overlays a code module bar on sites that run a code module on that subsystem.

Hover over a bar in the graph to display a tooltip with the following information:

- Plot Name
- Step Name
- Number of Sites
- Batch Index
- Site
- Subsystem

- Code Module Time or Value (When you select **Compare Data Sets** in the **Mode** section of the analyzer, the value in the tooltip includes the value of the bar the mouse is hovering over, the matching value in the other log file, and the difference between the two values.)

Click the back button next to the title in the upper left corner of the graph to view the Step Times Statistics graph.

Scatter Distribution Graph

Select **Scatter Distribution** in the **Graph Type** control to display a graph you can use to view the statistical distribution of test times. Use this view to identify outliers in test time distribution. Select **Socket Time** to view the distribution of your entire test program. Alternatively, select **Code Module Time** in the **Timing Method** section, and then select **MainSequence Time** to view the distribution of the entire test program. Select **Single Step Time** and specify a step name to view the distribution of a single step's test times.

Each plot represents a given number of sites. Each point represents the average test time over all sites in the configuration on the given batch index. Click a point on the Scatter Distribution graph to go to that batch index in the Single Step graph to view more detail on individual sites. Click the back button next to the title in the upper left corner of the graph to return to the Scatter Distribution graph.

Hover over a point in the graph to display a tooltip with the following information:

- Plot Name
- Batch Index
- Number of Sites
- Value

Click the back button next to the title in the upper left corner of the graph to return to the Step Times Statistics graph.

Socket Times/PTE (Parallel Test Efficiency) Graph

Select **Socket Times/PTE** in the **Graph Type** control to display a graph you can use to view the overall socket time or `MainSequence` time of the test program for different multisite configurations.

The bars on the graph display the test time, and the diamonds on the graph display the calculated PTE of the test time relative to the single site configuration test time. TSM uses the following formula to calculate the PTE:

$$PTE(n) = \left[1 - \frac{T_n - T_1}{T_1 \times (n-1)}\right] \times 100\%$$

n = *number of sites*, T_1 = single site test time, T_n = n site test time

Use the following options in the **Target Socket or MainSequence Time** section of the analyzer to configure an optional plot that displays the target test time plot for each multisite configuration. The graph displays a horizontal line on each bar to indicate the time under which the target PTE percentage has been met.

- **Target PTE(%)**—Specify the target PTE value.
- **Benchmark Single Site Socket or MainSequence Time (ms)**—Specify the single site socket or MainSequence time in milliseconds to use to calculate the target test time.

Hover over a bar in the graph to display a tooltip with the following information:

- Plot Name
- Value (When you select **Compare Data Sets** in the **Mode** section of the analyzer, the value in the tooltip includes the value of the bar the mouse is hovering over, the matching value in the other log file, and the difference between the two values.)

Timing Method

The **Timing Method** section of the analyzer includes the following options:

- **Total Time**—Total time to run the step, including any TestStand overhead and the time it takes to run any code modules.

- **Code Module Time**—Only the time it takes to run the code module, if any, for the step.

When you select **Socket Times/PTE** in the **Graph Type** control of the analyzer, the **Timing Method** section includes the following options:

- **Socket Time**—Total time to run the `MainSequence` sequence, including any TestStand overhead. Socket time is different from cycle time, which includes the socket time and the time required to perform other tasks, such as binning, placing DUTs, and generating reports.
- **MainSequence**—Only the time it takes to run the `MainSequence` sequence.

Sorting Method

The **Sorting Method** section of the analyzer for the Step Times Statistics Graph includes the following options based on whether you select **Single Data Set** or **Compare Data Sets** in the **Mode** section:

- **Execution Order**—Sorts the steps in execution order. This is the default setting for the Sorting Method option.

Note If you are comparing data sets and the execution order differs between the two sets, the analyzer uses the execution order of the base log file data. If the execution order changes after the first time you take a test program performance measurement, the analyzer reflects the execution order used in the first run.

- **Step Name**—Sorts the steps alphabetically by step name.
- **Test Time (Base, Modified, Difference)**—Sorts steps by the selected test time statistic, highest to lowest. You can sort steps by the average, minimum, lower quartile, median, upper quartile, or maximum of their test times. For multiple sites, the analyzer sorts by the time of the highest number of sites. If you are comparing data sets, you can sort by the test times in the base log file data, the modified log file data, or the difference between the two.

- **Show Top X Test Times**—Specifies the number of steps with the highest test times to display on the graph.
- **Show All Test Times**—Displays all the steps with the highest test times.

- **Sort by PTE (Base, Modified)**—Sorts steps by the worst PTE value. If you are comparing data sets, you can sort by the PTE values in the base log file data or the modified log file data.

Show Single Site Data

The analyzer displays this option only when performance data contains only data for single site and one other multisite configuration. In this situation, the analyzer hides the single site data by default because it typically is being included only for the purpose of calculating PTE. Enabling this option overrides this behavior and displays the single site data in the graph.

Test Program Performance Analyzer Filters (TSM)

Click the **Filters** tab at the bottom of the Test Program Performance Analyzer to expand or collapse the tab, which contains the following data items to display or hide on the analyzer graph. By default, the analyzer graph displays all the data items.

- Number of sites—The number of sites in the site configuration on which the step was run.
- Site—The site on which the step was run.
- Status
- Batch #
- Sequence file
- Sequence
- Step type
- Step group
- Step name
- Step ID

To exclude items, select one or more the items in the **Included** section of the Filters tab and click the single left arrow. Click the double left arrow to exclude all items in the currently selected filter.

To include items, select one or more items in the **Excluded** section of the Filters tab and click the single right arrow. Click the double right arrow to include all items in the currently selected filter.

Use the search box below the **Included** section and the **Excluded** section to search for items the filter category contains.

Each filter displays only the items that exist in the currently filtered data set. Excluding items in one filter might also hide the items in another filter, which allows you to narrow down the data set using multiple criteria.

Use the following buttons to complete the corresponding actions:

- **Reset All Filters**—Restores the default filter settings for the data sets the graph displays. By default, the analyzer graph displays all the data items.
- **Load**—Loads filter settings from a previously saved filter setting file (`.filtersettings`).
- **Save**—Saves the current filter settings to a filter setting file (`.filtersettings`).

Comparing Data in the Test Program Performance Analyzer (TSM)

Select **Compare Data Sets** in the **Mode** section of the Test Program Performance Analyzer to display two data set log files in the analyzer graph. Use the **Base Log File Path** and **Modified Log File Path** controls to select the data set log files to compare. If the **Modified Log File Path** control is empty when you change display modes, the analyzer launches a browse dialog box for you to select the file you want to use as the modified log file to compare against the existing base log file.

The analyzer graph displays the data in the following colors to indicate differences:

- **Gray**—Base data values.

- **Green**—A modified data value that is less than the base data value indicates a performance increase.
- **Red**—A modified data value that is greater than the base data value indicates a performance decrease.
- **Blue**—A modified data value that has a difference in performance from the base data value that is less than the percentage value specified in the **Highlight Differences Above** control in the **Mode** section of the analyzer.

By default, the analyzer highlights differences only when the difference between the modified data value and the base data value is greater than 1%. Use the **Highlight Differences Above** control in the **Mode** section of the analyzer to change this threshold.

When you complete a test program performance measurement while the analyzer is already comparing two data sets, TSM prompts you to compare the newly generated data set to the current base data set displayed in the analyzer or to the current modified data set displayed in the analyzer.

Log Browser Window

Click the **Log Browser** button near the upper right corner of the Test Program Performance Analyzer to open the Log Browser window, which you can use to view or compare multiple log files that represent different modifications of a test program.

Complete the following steps to view or compare log files.

1. In the **Log Directory** path control, select the top-level directory that contains the log files you want to view to compare. The column on the left displays relative subdirectories.
2. Complete the following steps to view file(s) in Single Data Set mode or Compare Data Sets mode.
    a. **Single Data Set** – Double-click the row of the file you want to load. The background row color turns blue.

b. **Compare Data Sets** – Right-click the row of the base log file you want to load and chose **Select Base Log File** from the context menu. The background row color of the base log file you selected turns gray. Right-click the row of the modified log file you want to load and chose **Select Modified Log File** from the context menu. The background row color of the modified log file you selected turns blue.

Test Program Performance Measurement Data Logs (TSM)

When you measure test program performance, TSM generates a performance log file with the data gathered during the series of test executions and a summary log file that describes the testing environment.

The Test Program Performance Analyzer uses the performance log file to analyze and display the performance measurement data.

Performance Log File

The performance log file contains measurement data for each execution in the performance measurement operation. The performance log file stores data in a comma-separated values (`.csv`) format and includes the following information about each step in the test program execution.

Column Name	Notes
NumberOfSites	Use this information to distinguish among the site configurations the log file includes.
Site	
SequenceFile	Blank for entries in the log file that correspond to the steps in the process model, such as the MainSequence Callback step.
Sequence	Blank for entries in the log file that correspond to the steps in the process model, such as the MainSequence Callback step.
StepGroup	Setup, Main, or Cleanup
StepName	

StepId	
StepType	
SitesTested	String that contains a comma-separated list of sites on which the code module executed. This value corresponds to the subsystem information on the Options tab of a Semiconductor Action or Multi Test step.
LoopIndex	1-indexed number of the batches tested during the execution.
ModuleTime	Execution time for the code module associated with the step. This value is `0` if the step does not call a code module.
TotalTime	Total execution time for the step, including the module time if the step calls a code module. For Sequence Call steps, includes the execution time of the entire sequence called from the step.
Status	`Result.Status` for each step, such as `Passed`, `Failed`, `Done`, or `Error`.
SocketTime	Total execution time of the main sequence. This value is logged only once for each loop of the main sequence and is left blank for rows that correspond to individual steps.

Summary Log File

The summary log file contains information about the testing environment. The summary log file stores data in a plain text (.txt) format and includes the following information:

- Test Program
- Active Configuration
- Offline Mode
- Process Model Path
- Operator Interface
- Operator Interface Arguments
- LabVIEW Adapter Server

- Enabled Result Processors
- Notes
- TestStand Version
- Number of Sites

Saving Data in the Test Program Performance Analyzer (TSM)

You can export the data set(s) the Test Program Performance Analyzer displays as an image or as a comma-separated values file.

Click the **Export Graph to .PNG** button to export the current graph display as a `.png` file.

Click the **Export Log Files(s)** button to export the filtered data set the current graph displays to a new `.csv` file. If you are comparing data sets in the analyzer, TSM creates two `.csv` files of the filtered data the graph displays.

Perform Analysis in Parallel with Measurements (TSM)

If a test program spends significant time analyzing measurement data, you might be able to improve performance by executing the analysis portion of the test in parallel with the measurement portion of the test.

See Also

Asynchronous Analysis Example

Performing Inline Quality Assurance Testing (TSM)

A tester might need to perform periodic or random quality assurance (QA) tests to identify errors or unintended behavior in some component of the test system. Typically, a QA test specifies limits that are wider than those of a normal test. Because an actual test might pass with a value that is very close to a limit, wider

limits ensure that an inline QA failure occurs only when a problem exists with the tester.

A technician or operator might run a dedicated QA test—separate from a standard test sequence—on a tester as part of the manufacturing process or to perform an audit on the test system. However, in some cases, the test engineer might want to perform inline QA testing, in which one or more QA tests exists within a standard test sequence.

TSM Implementation

Use a station setting TSM provides to enable or disable inline QA testing functionality for the test station. When you enable inline QA testing for the test station, you must specify an inline QA algorithm that specifies the inline QA testing behavior. In the test program main sequence file, insert an Inline QA Test Block step to specify a block of steps that perform inline QA testing only under certain conditions. TSM maintains a queue of inline QA states, where each state is a Boolean value that indicates whether to perform inline QA for a DUT.

The steps within the inline QA test block execute only when all of the following conditions are true:

- The StationSettings.Standard.InlineQAEnabled Boolean property is `True`.
- The `Step.ConditionExpr` property of the Inline QA Test Block step specifies an expression that evaluates to `True`.
- The next inline QA state, which is the next Boolean value removed from the TSM queue of inline QA states, is `True`.

Note If multiple inline QA test blocks exist in the sequence, only the first one that executes dequeues the inline QA state for the DUT. Subsequent inline QA test blocks use the inline QA state that the first inline QA test block dequeued.

Complete the following steps to implement inline QA testing functionality for a test program.

1. Create an inline QA algorithm sequence file and edit the GetNextInlineQAStateQueueItems callback sequence.
2. Enable inline QA on the test station and specify the inline QA algorithm sequence file you created in step 1 as the inline QA algorithm sequence file to use for the test station.
3. Insert a single Inline QA Test Block step instance in the `MainSequence` sequence of the test program main sequence file.
4. Insert additional steps within the Inline QA Test Block that call code modules that perform the inline QA tests.

Notes

- The default expression for the `Step.ConditionExpr` property evaluates to `True` when the `MainSequence` sequence has not yet encountered a sequence failure. You can modify the expression the `Step.ConditionExpr` property specifies to customize the condition for which the inline QA test block executes.
- You can place multiple Inline QA Test Block step type instances at any location in any test sequence, but NI recommends that you place only a single instance after all standard test steps in the `MainSequence` sequence of the test program main sequence file.

Creating an Inline QA Algorithm Sequence File (TSM)

Complete the following steps to use the `InlineQAAlgorithm.seq` file, located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory, as a starting point for an inline QA algorithm sequence file you create.

1. Copy the `InlineQAAlgorithm.seq` file from the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory

to the <TestStand Public>\Components\Modules\NI_SemiconductorModule\InlineQA directory.

2. Rename the <TestStand Public>\Components\Modules\NI_SemiconductorModule\InlineQA\InlineQAAlgorithm.seq file using the **<CompanyName>_<InlineQAAlgorithmName>**.seq convention.
3. Open the inline QA algorithm sequence file.
4. Add steps to the GetNextInlineQAStateQueueItems callback sequence to populate the **NextInlineQAStateQueueItems** sequence parameter with a queue of inline QA states.
5. Save the inline QA algorithm sequence file.

See Also

Enabling Inline QA on the Test Station

GetNextInlineQAStateQueueItems Callback (TSM)

TSM calls the GetNextInlineQAStateQueueItems callback sequence in the inline QA algorithm sequence file specified for the test station.

The GetNextInlineQAStateQueueItems callback sequence specifies a queue of inline QA states, where each state represents an enabled or disabled state that indicates whether to perform inline QA for a particular DUT. When the queue does not contain enough inline QA states for the next set of DUTs to test, TSM calls the GetNextInlineQAStateQueueItems callback sequence to obtain additional inline QA states.

The GetNextInlineQAStateQueueItems callback sequence accepts the following parameters:

- **MinimumInlineQAStateQueueItemsRequired** [In]—The minimum number of inline QA states the GetNextInlineQAStateQueueItems callback sequence must return for TSM to successfully perform inline QA testing. To improve performance, return more than the minimum number of required states this parameter specifies to limit the number of times TSM calls the GetNextInlineQAStateQueueItems callback sequence.

- **LotSettings** [In]—An instance of the NI_SemiconductorModule_LotSettings data type.
- **StationSettings** [In]—An instance of the NI_SemiconductorModule_StationSettings data type.
- **NextInlineQAStateQueueItems** [Out]—Array of Boolean values, where each Boolean value corresponds to an inline QA state. The `GetNextInlineQAStateQueueItems` callback sequence must resize this array and populate it with the next inline QA state for the TSM queue of inline QA states. The number of inline QA states populated must be greater than or equal to the number of required inline QA states the **MinimumInlineQAStateQueueItemsRequired** parameter specifies.

Enabling Inline QA on the Test Station (TSM)

Use the ConfigureStationSettings or GetStationSettings callback sequences to set the StationSettings.Standard.InlineQAEnabled Boolean property to `True` or `False` to enable or disable inline QA on the test station.

The General tab of the default Configure Station Settings dialog box contains an **Enable Inline QA** option that also sets the value of the `StationSettings.Standard.InlineQAEnabled` property.

When you enable inline QA on the test station, you must set the value of the StationSettings.Standard.InlineQAAlgorithmSequenceFilePath property to the absolute path of the inline QA algorithm sequence file you want to use on the test station. You can also use the **Inline QA Algorithm** control in the default Configure Station Settings dialog box to specify the inline QA algorithm sequence file.

Part Average Testing (TSM)

Part average testing (PAT) is a method based on statistical analysis to identify and fail parts that have characteristics significantly outside the normal distribution of other parts in the same lot. The substantial difference of these parts, which might still fall within the test program limits, could indicate the potential for early part failure.

Generally, part average testing collects data from previously tested parts and compares each measurement for the current part to the mean of the previous measurements. If the measurements for the current part are outside a certain number of standard deviations from the mean, the part fails.

Refer to the **Guidelines for Part Average Testing** document (AEC - Q001 Rev-D version) published by the Automotive Electronics Council for more information about part average testing.

TSM Implementation

TSM does not install a default implementation of part average testing. You must use the TSM PAT plug-in architecture to customize and perform part average testing with TSM. TSM PAT plug-ins include a required PAT callback sequence file and corresponding code modules. The PAT callback sequence file contains PAT entry point sequences that TSM calls during execution to accomplish part average testing. Use the example PAT plug-in, located in the `<TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In` directory, as a starting point for custom PAT plug-ins you create.

Use the PAT Algorithm Settings panel of the Test Program Editor to edit settings the PAT callback sequence file defines to customize the behavior of the algorithm execution for each test program.

Use the Part Average Testing tab of the Semiconductor Multi Test step to enable and configure part average testing for individual tests in a test program. The PAT environment settings determine which settings to display in the Part Average Testing tab.

Refer to the Part Average Testing Examples for information about enabling and performing part average testing (PAT) in a test program.

Execution

The first time a PAT-enabled test executes, TSM automatically generates one or more additional PAT tests associated with the original test. After executing all tests in the `MainSequence` sequence of the test program, TSM calls the PAT entry point sequences to customize and perform the PAT tests. TSM appends PAT test results to

the MainSequence sequence test results so that PAT test results appear in the data logs and reports.

Alternatively, you can insert one or more Perform Part Average Testing steps in your test program to perform the PAT tests at specific points during program execution. Use the result from PAT tests performed by these steps to control the execution flow of the program. Tests that are performed by a Perform Part Average Testing step are not performed after the MainSequence sequence of the test program.

Static Limits File

Use the IPartAverageTestingStaticLimitLoader.LoadStaticLimits method in the TSM Application API to load a static limits file that uses the same structure as a TSM test limits file. To read and use limits from a different file format, you must implement a custom file reader and a custom data structure to store the limits.

See Also

Part Average Testing Examples

Part Average Testing Plug-In Architecture (TSM)

Use the example PAT plug-in, located in the <TestStand Public>\Examples\NI_SemiconductorModule\Part Average Testing\Example Part Average Testing Plug-In directory, as a starting point for custom PAT plug-ins you create. Refer to the Part Average Testing Examples for information about enabling and performing part average testing (PAT) in a test program.

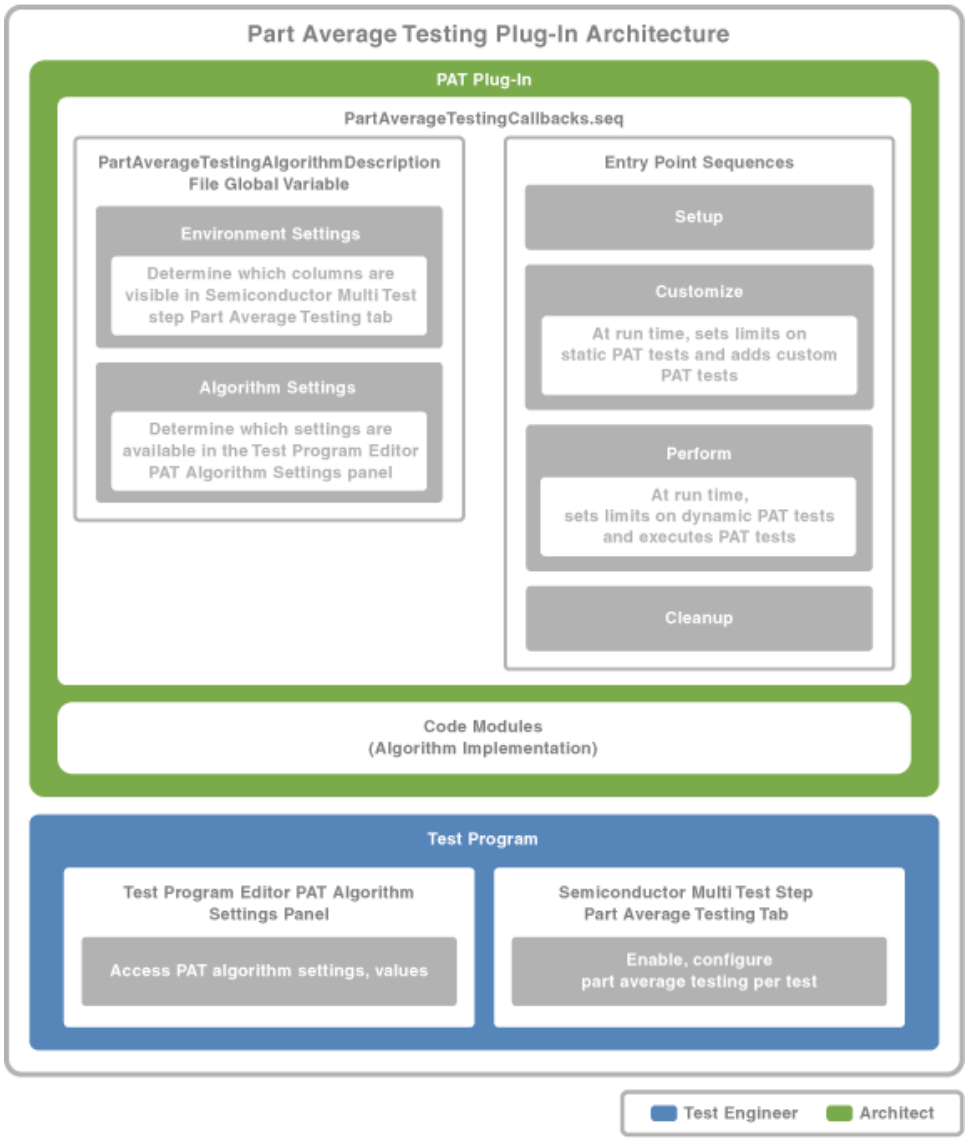
You must create and deploy the following plug-in files to implement custom part average testing (PAT) for a test program:

- A PAT callback sequence file that meets the following requirements:
  - The filename must be PartAverageTestingCallbacks.seq.
  - The file must reside in the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory.

- The file must include a sequence file global variable named `PartAverageTestingAlgorithmDescription` that is an instance of the NI_SemiconductorModule_PATAlgorithmDescription data type to describe the environment settings and algorithm settings.
- The file must implement each of the PAT entry point sequences TSM calls at specific points during test program execution.

- Code modules that implement specific PAT algorithm tasks and settings.

The following graphic illustrates the PAT plug-in architecture



See Also

Part Average Testing Examples

Part Average Testing Environment Settings (TSM)

The part average testing (PAT) environment settings define characteristics of the PAT algorithm TSM uses to customize the TSM environment for the PAT algorithm. Specify these settings in the following properties in the `FileGlobals.PartAverageTestingAlgorithmDescription.EnvironmentSettings` container property:

- **TestNumberOffsetForPinGroupTests**—Specifies the offset that the Semiconductor Multi Test step adds to the **Dynamic PAT Test Number**, to the **Static PAT Test Number**, and to the **Base PAT Test Number** when the step creates additional tests for pins in a pin group. When a test specifies a pin group in the **Pin** column in the Tests table on the Tests tab, the Tests table inserts a test for each pin in the pin group. The **Test Number** column is computed by adding the test number specified for the pin group test to the zero-based index of the pin in the pin group. Although the Part Average Testing tab does not display the inserted tests for pin group tests, if part average testing is enabled for a test that specifies a pin group, the step creates PAT tests for each inserted test. The values for the Dynamic PAT Test Number, Static PAT Test Number, and Base PAT Test Number are computed by adding the test number specified by the pin group test to the product of the zero-based index of the pin in the pin group and the value of `TestNumberOffsetForPinGroupTests`.
- `AllowExecutionWithDefaultSettingValues`—Indicates whether TSM executes a test program that does not include any PAT algorithm settings. TSM uses this setting only when no PAT algorithm settings exist in the test program. When this setting is `True`, TSM uses the default values for PAT algorithm settings stored in the `FileGlobals.PartAverageTestingAlgorithmDescription.AlgorithmSettings` property in the PAT callback sequence file. When this setting is `False`, TSM reports a run-time error.

- `EnablePerformPartAverageTestingStep`—Enables the Perform Part Average Testing step in the PAT algorithm. By default, this setting is `False` and the Perform Part Average Testing step is disabled.
- `StepSettingsPaneUI`—Contains Boolean properties that specify which of the following columns to display on the Part Average Testing tab of the Semiconductor Multi Test step.
  - `EnableBaseTestNumberColumn`—Specifies whether the **PAT Base Test Number** column is visible.
  - `EnableDynamicEnableColumn`—Specifies whether the **Enable Dynamic PAT** column is visible.
  - `EnableDynamicTestNumberColumn`—Specifies whether the **Dynamic PAT Test Number** column is visible.
  - `EnableDynamicTestNameColumn`—Specifies whether the **Dynamic PAT Test Name** column is visible.
  - `EnableDynamicSoftwareBinColumn`—Specifies whether the **Dynamic PAT Software Bin** column is visible.
  - `EnableDynamicLowLimitColumn`—Specifies whether the **Dynamic PAT Low Limit** column is visible.
  - `EnableDynamicHighLimitColumn`—Specifies whether the **Dynamic PAT High Limit** column is visible.
  - `EnableStaticEnableColumn`—Specifies whether the **Enable Static PAT** column is visible.
  - `EnableStaticTestNumberColumn`—Specifies whether the **Static PAT Test Number** column is visible.
  - `EnableStaticTestNameColumn`—Specifies whether the **Static PAT Test Name** column is visible.
  - `EnableStaticSoftwareBinColumn`—Specifies whether the **Static PAT Software Bin** column is visible.

See Also

Part Average Testing Examples

Part Average Testing Algorithm Settings

Part Average Testing Algorithm Settings (TSM)

The part average testing (PAT) algorithm settings are custom settings that define the behavior of a specific PAT algorithm. By default, a PAT algorithm does not include algorithm settings. You must create each setting for a PAT algorithm, and you must assign default values to each setting.

Creating PAT Algorithm Settings

To create a PAT algorithm setting, add string, number, Boolean, and enumeration properties to the `FileGlobals.PartAverageTestingDescription.AlgorithmSettings` container property. To further refine how TSM displays the setting in the PAT Algorithm Settings panel of the Test Program Editor, add an attribute to the setting property that is an instance of the NI_SemiconductorModule_PATAlgorithmSettingUIAttribute data type and name the attribute `UI`. Set the following properties in the `UI` container to control how TSM treats the setting:

- `DisplayName`—Specifies the text to display in the **Name** column for the setting. If this property is an empty string, TSM uses the algorithm setting property name.
- `Description`—Specifies an additional description of the setting that the PAT Algorithm Settings panel displays as a tooltip for the setting.
- `BrowseButtonOption`—Specifies whether TSM displays a browse button in the **Value** column for finding files or directories. Use this property to indicate whether the setting is a file or a directory.
- `RequireAbsolutePath`—Specifies whether TSM inserts absolute paths by default when browsing for a file or directory. TSM uses this property only if the **BrowseButtonOption** is set to `Display file browse button` or `Display directory file browse button`.

- `MinimumNumberValue`—Specifies the minimum value for a Number setting. This property has a default value of `-Inf`.
- `MaximumNumberValue`—Specifies the maximum value for a Number setting. This property has a default value of `+Inf`.
- `DisplaySoftwareBinComboBox`—Specifies that the Number setting corresponds to a bin number. TSM displays a combobox with the list of software bins for the current test program in the **Value** column. Bins must be defined in the bin definitions file.

Note If you change the PAT algorithm settings in the `PartAverageTestingAlgorithmDescription` file global variable of the PAT callback sequence file (`PartAverageTestingCallbacks.seq`), you must update existing test programs that contain PAT algorithm settings to use the latest settings. Click the **Update Test Program Settings** button in the PAT Algorithm Settings panel of the Test Program Editor to update the PAT algorithm settings in a test program to match the current PAT algorithm settings. TSM reports a run-time error when you execute a test program that includes PAT algorithm settings that do not match the settings in the PAT callback sequence file.

Assigning Values to PAT Algorithm Settings

Use the PAT Algorithm Settings panel of the Test Program Editor to set values for the PAT algorithm setting so that each test program can use unique values.

Obtaining PAT Algorithm Settings at Run Time

Use the `PartAverageTestingAlgorithmSettings` property on the **SetupPartAverageTestingCallbackArgs** parameter of the PAT Setup entry point sequence to obtain the values of PAT algorithm settings at run time. Refer to the `IPartAverageTestingAlgorithmSettings` interface in the TSM Application API for more information.

See Also

Part Average Testing Examples

Part Average Testing Environment Settings

Part Average Testing Entry Points (TSM)

At specific points during test program execution, TSM calls the following set of required part average testing (PAT) entry point sequences in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\PartAverageTestingCallbacks.seq` file. TSM calls the entry point sequences for each site independently.

- Setup—Initializes the PAT algorithm for the site.
- Customize—Customizes all the PAT tests for the site.
- Perform—Performs part average testing for the site by setting limits and evaluating PAT tests.
- Cleanup—Performs finalization tasks.

See Also

Part Average Testing Examples

Part Average Testing Setup Entry Point (TSM)

Use the part average testing (PAT) Setup entry point to perform initialization tasks for a single site. TSM calls the entry point for each site when starting a new lot. The Setup entry point typically obtains values of PAT algorithm settings from the test program, obtains limits for static PAT tests, and initializes data structures for computing limits for dynamic PAT tests.

Parameters

The Setup entry point accepts the following parameters:

- **ModelPluginConfiguration** [In]—An instance of the NI_ModelPluginConfiguration data type that contains the configuration and run-time variables for the set of all active process model plug-in instances. Use this parameter to extract information, such as report paths and directories.
- **ModelData** [In]—Contains information about the process model used for the lot testing.
- **SetupPartAverageTestingCallbackArgs** [In]—A reference to an object that implements the `ISetupPartAverageTestingCallbackArgs` interface in the TSM Application API. This interface provides properties to obtain values of PAT algorithm settings, the site index, and other information.
- **PartAverageTestingRuntimeData** [Out]—An object reference that stores run-time data, such as test statistics, to access in the other entry points. TSM passes a reference to this object to each of the other PAT entry points.

See Also

Part Average Testing Examples

Part Average Testing Customize Entry Point (TSM)

Use the part average testing (PAT) Customize entry point to customize PAT tests before the PAT tests execute on a site. TSM calls this entry point if a PAT-enabled test is executed for the first time while testing the current DUT. TSM calls this entry point after executing the `MainSequence` sequence or when executing a Perform Part Average Testing step. TSM passes the PAT-enabled tests and their associated PAT tests to the entry point sequence.

Some typical customizations the entry point performs include setting limits on static PAT tests, setting PAT test numbers and test names, and creating PAT tests associated with a PAT-enabled test.

Parameters

The Customize entry point accepts the following parameters:

- **ModelPluginConfiguration** [In]—An instance of the NI_ModelPluginConfiguration data type that contains the configuration and run-time variables for the set of all active process model plug-in instances. Use this parameter to extract information, such as report paths and directories.
- **ModelData** [In]—Contains information about the process model used for the lot testing.
- **PartAverageTestingRuntimeData** [In]—The object reference created in the Setup entry point that stores run-time data, such as test statistics.
- **CustomizePartAverageTestingCallbackArgs** [In]—A reference to an object that implements the `ICustomizePartAverageTestingCallbackArgs` interface in the TSM Application API. This interface provides the list of PAT-enabled tests and associated PAT tests to customize. The tests this object references include only the tests that executed for the first time while testing the current DUT.

See Also

Part Average Testing Examples

Part Average Testing Perform Entry Point (TSM)

Use the part average testing (PAT) Perform entry point to perform the PAT tests after all the tests in the `MainSequence` sequence have executed on a site or during the execution of a Perform Part Average Testing step. TSM calls this entry point if any PAT-enabled tests executed while testing the current DUT. The Perform entry point sets the limits on dynamic PAT tests and executes each dynamic and static PAT test.

Parameters

The Perform entry point accepts the following parameters:

- **ModelPluginConfiguration** [In]—An instance of the NI_ModelPluginConfiguration data type that contains the configuration and run-time variables for the set of all active process model plug-in instances. Use this parameter to extract information, such as report paths and directories.
- **ModelData** [In]—Contains information about the process model used for the lot testing.
- **PartAverageTestingRuntimeData** [In]—The object reference created in the Setup entry point that stores run-time data, such as test statistics.
- **CustomizePartAverageTestingCallbackArgs** [In]—A reference to an object that implements the `ICustomizePartAverageTestingCallbackArgs` interface in the TSM Application API. This interface provides the list of PAT tests to execute.

See Also

Part Average Testing Examples

Part Average Testing Cleanup Entry Point (TSM)

Use the part average testing (PAT) Cleanup entry point to perform finalization tasks and to dispose of data structures.

Parameters

The Cleanup entry point accepts the following parameters:

- **ModelPluginConfiguration** [In]—An instance of the NI_ModelPluginConfiguration data type that contains the configuration and run-time variables for the set of all active process model plug-in instances. Use this parameter to extract information, such as report paths and directories.
- **ModelData** [In]—Contains information about the process model used for the lot testing.

- **PartAverageTestingRuntimeData** [In]—The object reference created in the Setup entry point that stores run-time data, such as test statistics.

See Also

Part Average Testing Examples

Specifying Settings for the Current Test Station (TSM)

You can specify test station configuration options for the tester, such as handler configuration or data logging preferences, that apply to all test lots and that persist during restart and shutdown operations. The test engineer or technician usually configures station settings when initially setting up or later reconfiguring a tester. Generally, an operator has only restricted access to station settings to ensure that critical configuration options do not change.

The test program can use station information to determine how to execute tests. For example, station settings might specify the type of handler to use with the test program, or whether the test station performs inline quality assurance testing. When a test station is reconfigured, such as to specify a different handler or to change the functionality of the tester, the station settings must be updated to account for the changes.

TSM Implementation

The NI_SemiconductorModule_StationSettings data type includes properties that correspond to some fields of the Master Information Record (MIR) and Site Description Record (SDR) of version 4 of the Standard Test Data Format (STDF) and includes other properties specific to TSM. You can access the station settings from the test program, and you can modify the default TSM station settings callbacks to customize how TSM obtains the settings.

The **Configure Station** button in the default TSM operator interface, the **Semiconductor Module»Configure Station** menu item in the TestStand Sequence Editor, and the **Configure Station** button on the TSM toolbar call the ConfigureStationSettings callback sequence to obtain the settings for the current test station. Use the `ConfigureStationSettings` callback sequence to

prompt a test engineer or technician to manually configure station settings in the Configure Station Settings dialog box or to use another mechanism that requires user input.

Note The default TSM operator interface restricts access to the **Configure Station** button to users with the ConfigApp privilege.

Customizing the Behavior for Obtaining Station Settings (TSM)

TSM obtains station settings by calling the following callback sequences, located in the <TestStand>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file:

- ConfigureStationSettings—The **Configure Station** button in the default TSM operator interface or the **Semiconductor Module»Configure Station** menu item in the TestStand Sequence Editor calls the `ConfigureStationSettings` callback sequence to obtain the settings for the current test station. Use the `ConfigureStationSettings` callback sequence to prompt a test engineer or technician to manually configure station settings in a dialog box or to use another mechanism that requires user input. The default implementation of the `ConfigureStationSettings` callback sequence launches the default Configure Station Settings dialog box.
- GetStationSettings—Use the `GetStationSettings` callback sequence to programmatically obtain station settings when the test program begins executing without requiring much, if any, test engineer or technician interaction. For example, you can specify that the callback sequence queries a database for the station information related to a barcode or DUT type. The default implementation of the `GetStationSettings` callback sequence does not perform any operations.

Complete the following steps to override the default `ConfigureStationSettings` or `GetStationSettings` callback sequence and customize the behavior for obtaining station settings.

1. Determine whether a sequence file named `SemiconductorModuleCallbacks.seq` exists in the <TestStand Public>\Components\Callbac

ks\NI_SemiconductorModule directory. If the sequence file does not exist, create it and ensure that it does not contain any sequences.

2. Copy the ConfigureStationSettings or GetStationSettings callback sequences from the <TestStand>\Components\Modules\NI_SemiconductorModule\Templates\SemiconductorModuleCallbacks.seq file to the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file, and make changes to the copy.
3. To customize the default Configure Station Settings dialog box in the ConfigureStationSettings callback sequence, complete the following additional steps.
    a. Copy the Display Configure Station Settings Dialog step from the ConfigureStationSettings callback sequence of the <TestStand>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file to the ConfigureStationSettings callback sequence of the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file.
    b. Copy the contents of <TestStand>\Components\Callbacks\NI_SemiconductorModule\StationSettingsDialogs to <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\StationSettingsDialogs and make changes to the copy of the LabVIEW project. If you are making extensive customizations to the LabVIEW project, use the Debugging the Station Settings dialog box to enable automatic testing when you makes changes to the station settings.
    c. Rebuild the packed project library build specification in the project to update the copy of the LabVIEW packed project library.

See Also

Customizing the Behavior for Obtaining Lot Settings

Debugging the Station Settings Dialog Box (TSM)

If you customize the Station Settings dialog box, you can use the following steps to automatically test new changes you make to the station settings. With these settings configured, TestStand Semiconductor Module (TSM) calls the VIs directly when configuring the station settings so any changes you make are reflected immediately. You do not have to rebuild the packed project library.

1. Complete the following steps to configure TSM to call the Station Settings dialog box source VIs instead of a packed project library.
    a. In the TestStand Sequence Editor, select **Configure»Adapters** and select **LabVIEW**.
    b. Click **Configure** and select **LabVIEW Development System**.
    c. Click **OK** in the LabVIEW Adapter Configuration dialog box.
    d. Click **Done** to close the Adapter Configuration dialog box.
    e. Open the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` file.
    f. In the Display Configure Station Settings Dialog step of the `Configure StationSettings` sequence, click the **Module** tab.
    g. Click **Advanced Settings** and remove the checkmark from the **Always Run VI in LabVIEW Run-Time Engine** checkbox.
    h. Click **Close** to close the LabVIEW Advanced Settings dialog box.
    i. On the **Module** tab, change the VI path from the default value of `StationSettingsDialogs.lvlibp\Configure Station Settings.vi` to `Configure Station Settings.vi`.
    j. Save the sequence file.
2. Complete the following steps to configure the Station Settings dialog box VIs for debugging.
    a. Open the Configure Station Settings VI.
    b. Select **File»VI Properties** to open the VI Properties dialog box.

c. In the **Category** drop-down menu, select **Window Appearance**.
d. Click **Customize** to open the Customize Window Appearance dialog box. View and record the current custom settings so you can restore these settings when you finish debugging.
e. Close the Customize Window Appearance dialog box.
f. In the VI Properties dialog box, select **Default**.
g. Click **OK** to close the VI Properties dialog box.
h. Save the VI.

3. Debug the Station Settings dialog box VIs.
4. When you finish debugging the VIs, complete the following steps to restore the settings to the previous state.
    a. Open the Configure Station Settings VI.
    b. Select **File»VI Properties** to open the VI Properties dialog box.
    c. In the **Category** drop-down menu, select **Window Appearance**.
    d. Select **Custom** and then click **Customize** to launch the Customize Window Appearance dialog box.
    e. Restore the custom settings you recorded in step 2d.
    f. Click **OK** to close the Customize Window Appearance dialog box.
    g. Click **OK** to close the VI Properties dialog box.
    h. Save the VI.
5. Rebuild the packed project library specification as instructed in the Customizing the Behavior for Obtaining Station Settings topic to use the modified version of the dialog box with the restored settings.
6. Complete the following steps to restore TSM to call a packed project library instead of calling the VIs directly.
    a. In the Display Configure Station Settings Dialog step of the `Configure StationSettings` sequence, click the **Module** tab.
    b. Click **Advanced** settings and select the **Always Run VI in LabVIEW Run-Time Engine** checkbox.

c. On the **Module** tab, set the VI path to the `StationSettingsDialogs.lvlibp\Configure Station Settings.vi`.
d. Save the sequence file.

ConfigureStationSettings Callback (TSM)

The default implementation of the `ConfigureStationSettings` callback sequence launches the default Configure Station Settings dialog box, in which the test engineer or technician can configure the station settings. You can override this callback sequence to customize the behavior. You can also use the GetStationSettings callback sequence to programmatically obtain station settings when the test program begins executing without requiring much, if any, test engineer or technician interaction.

The `ConfigureStationSettings` callback sequence accepts the following parameters:

- **StationSettings** [In/Out]—An instance of the NI_SemiconductorModule_StationSettings data type. Configure the `ConfigureStationSettings` callback sequence to assign values for all the properties that you require.

  If you implement a dialog box for users to manually configure station information, ensure that the dialog box obtains values for all the properties of the **StationSettings** parameter that you require. If the default properties of the **StationSettings** parameter do not meet your requirements, you can add properties to the NI_SemiconductorModule_CustomStationSettings data type. The properties you add to the data type appear in the `Custom` container of the **StationSettings** parameter.

  TSM saves station settings to disk when the `ConfigureStationSettings` callback sequence sets the **Canceled** parameter to `False`.

- **Canceled** [Out]—Configure the `ConfigureStationSettings` callback sequence to set this parameter to `True` if the test engineer or technician cancels edits in the Configure Station Settings dialog box. TSM saves the

values of the **StationSettings** parameter to disk only when the **Canceled** parameter is `False`.

- **SemiconductorModuleManager** [In]—A reference to the Semiconductor Module Manager object that you use to call utility methods in the TSM Application API.

See Also

Station Settings from a Test Program

GetStationSettings Callback (TSM)

TSM calls the `GetStationSettings` callback sequence to programmatically obtain station settings without requiring much, if any, test engineer or technician interaction when execution begins.

The default implementation of the `GetStationSettings` callback sequence sets some standard station settings, such as `Standard.HandlerType`, `Standard.NodeName`, and `Standard.TesterType`. You can override this callback sequence to customize the behavior for when TSM attempts to determine station settings values at run time. You can also use the ConfigureStationSettings callback sequence to prompt a test engineer or technician to manually configure station settings in a dialog box or to use another mechanism that requires user input.

The `GetStationSettings` callback sequence accepts the following parameters:

- **StationSettings** [In/Out]—An instance of the NI_SemiconductorModule_StationSettings data type. Enter values for all the properties that you require.

  If you implement a mechanism for programmatically obtaining station information, ensure that the mechanism obtains values for all the properties of the **StationSettings** parameter that you require. If the default properties of the **StationSettings** parameter do not meet your requirements, you can add properties to the NI_SemiconductorModule_CustomStationSettings data type. The properties you add to the data type appear in the `Custom` container of the **StationSettings** parameter.

See Also

Accessing Station Settings from a Test Program

Accessing Station Settings from a Test Program (TSM)

Use the Get Test Information step to access standard and custom station settings in a test program sequence file.

Standard Station Settings

The following table lists the properties of the NI_SemiconductorModule_StandardStationSettings data type, the fields of the Master Information Record (MIR) or Site Description Record (SDR) of the Standard Test Data Format (STDF) version 4 specification that the STDF Log result processing plug-in sets using the properties, or other purpose of each property.

Note Use the Types window in the TestStand Sequence Editor to review properties on the standard type or to add properties to the custom type.

Property Name	Corresponding STDF Record Field or Other Purpose
AllowMorePinMapSitesThanTestSockets	Specifies whether TSM runs a test program with a pin map with more sites than the number of te st sockets configured in the model options. If yo u disable this option, TSM reports a run-time err or if the pin map has more sites than test socket s.
AllowFewerPinMapSitesThanTestSockets	Specifies whether TSM runs a test program with a pin map with fewer sites than the number of t est sockets configured in the model options. If y ou disable this option, TSM reports a run-time er ror if the pin map has fewer sites than test socke ts.
AvailableSiteNumbers	Specifies which site numbers from a pin map for a test program to use when running the test pro gram. Other sites in the pin map are ignored. Fo

	r example, you can set this property to disable specific sites or to use the particular connections of a pin map that match the DIB for the test station. If you leave this array empty, TSM uses the first **N** sites in the pin map, where **N** is the number of available test sockets in the station model up to the total number of sites in the pin map. The default Configure Lot Settings dialog box sets this property based on the Enabled Sites control.
DIBBoardId	SDR: DIB_ID
DIBBoardType	SDR: DIB_TYP
ExtraEquipmentId	SDR: EXTR_ID
ExtraEquipmentType	SDR: EXTR_TYP
GenerateUniquePartIds	Specifies whether TSM generates unique values for the PartId field of Part Results Records (PRRs) of the STDF log file. Set this value to `False` to generate unique PartId values using a custom algorithm you implement. When the `GenerateUniquePartIds` property is `True`, TSM reassigns the same unique Part ID to the part when it is retested. Complete the following steps to assign a new unique Part ID to a part when it is retested: 1. Using a text editor, open the `RuntimeSettings.ini.example` file located in the `<TestStand Application Data>\Cfg\NI_SemiconductorModule` directory. 2. Set the value of the GenerateUniquePartIdsForRetestedParts key to `true`. 3. Go to **File»Save as** and rename the modified file to `RuntimeSettings.ini`.
HandlerContactorId	SDR: CONT_ID
HandlerContactorType	SDR: CONT_TYP
HandlerDriverSequenceFilePath	Stores the absolute path and sequence filename of the real or simulated handler/prober driver.

	When the value of the `HandlerMode` property is `0`, TSM does not use this property.
HandlerId	SDR: HAND_ID
HandlerMode	Stores the current handler/prober mode.
HandlerType	SDR: HAND_TYP (Default value depends on the handler/prober mode. It is the handler/prober d river sequence filename or "No Handler" or "Si mulated Handler".)
InlineQAEnabled	Specifies whether you enabled inline QA.
InlineQAAlgorithmSequenceFilePath	Stores the absolute path of the inline QA algorit hm sequence file.
InterfaceCableId	SDR: CABL_ID
InterfaceCableType	SDR: CABL_TYP
LaserId	SDR: LASR_ID
LaserType	SDR: LASR_TYP
LoadBoardId	SDR: LOAD_ID
LoadBoardType	SDR: LOAD_TYP
NodeName	MIR: NODE_NAM (Default value is the TestStand Station ID.)
PartCountWindowSize	Specifies the number of part test results tracked per site. The site status indicators in the default operator interface display the results of the last **n** parts, where this option specifies **n**.
TimeStatisticsWindowSize	Specifies the number of parts used to compute t he average cycle time displayed in the default o perator interface. Use a negative value to use th e default, which is `10` ***n**, where **n** is the number of sites. Use a value of `0` to use the cycle time of all tested parts. Regardless of the setting value, TSM never includes the cycle time of the first pa rt in the average.
OfflineMode	Specifies whether Offline Mode is enabled, whic h allows you to develop, run, and debug test pro grams only on a computer without access to NI i nstruments.
ProbeCardId	SDR: CARD_ID
ProbeCardType	SDR: CARD_TYP

StationNumber	MIR: STAT_NUM
TesterSerialNumber	MIR: SERL_NUM
TesterType	MIR: TSTR_TYP (Default value is "National Instruments".)
TestFacilityId	MIR: FACIL_ID
TestFloorId	MIR: FLOOR_ID

See Also

Configuring Handler or Prober Support for a Test Program

NI_SemiconductorModule_StationSettings Data Type

The NI_SemiconductorModule_StationSettings data type defines the properties for each instance of the StationSettings sequence parameter, which the test program can access.

TSM obtains most of the values for the NI_SemiconductorModule_StationSettings data type from the ConfigureStationSettings callback sequence. The NI_SemiconductorModule_StationSettings data type contains the following properties:

- **Standard**—An instance of the NI_SemiconductorModule_StandardStationSettings data type. This property contains the standard station settings TSM recognizes.
- **Custom**—An instance of the NI_SemiconductorModule_CustomStationSettings data type. By default, this property is an empty container. You can add properties to the NI_SemiconductorModule_CustomStationSettings data type to add custom station settings that appear in this container.

See Also

GetStationSettings Callback

Accessing Station Settings from a Test Program

Get Test Information Step

Specifying Settings for the Current Lot under Test (TSM)

The production operator can specify the following information for the current lot of DUTs (test lot):

- Historical information, such as the wafer lot from which the DUTs came
- Descriptive information, such as DUT numbers or package types
- Conditions under which to test the DUTs, such as temperature, voltage, and so on

The test program can use lot information to determine how to execute tests. For example, lot settings might dictate which steps execute, what temperature to apply to a DUT, what voltage to use, and so on.

TSM Implementation

The NI_SemiconductorModule_LotSettings data type includes properties that correspond to some fields of the Master Information Record (MIR) of version 4 of the Standard Test Data Format (STDF) and includes other properties specific to TSM. You can access the lot settings from the test program, and you can modify the default TSM lot settings callbacks to customize how TSM obtains the settings.

The **Configure Lot** button in the default TSM operator interface, the **Semiconductor Module»Configure Lot** menu item in the TestStand Sequence Editor, and the **Configure Lot** button on the TSM toolbar call the ConfigureLotSettings callback sequence to obtain the settings for the current test lot. Use the `ConfigureLotSettings` callback sequence to prompt an operator to manually enter lot information in the Configure Lot Settings dialog box or to use another mechanism that requires user input.

Note The default TSM operator interface restricts access to the **Configure Lot** button to users with the Execute privilege.

See Also

Reports and Data Logs

Customizing the Behavior for Obtaining Lot Settings (TSM)

Use the following mechanisms to customize how TSM obtains lot settings for a test program.

- ConfigureLotSettings callback sequence—TSM calls the `ConfigureLotSettings` callback sequence when you trigger the ConfigureLotSettings command in the TSM operator interface or the TestStand Sequence Editor. If you set the `ConfigureLotWhenStartingLot` property on the SemiconductorModuleManager object using the TSM Application API to true, TSM also calls this callback sequence just before execution begins, when an operator initiates the start of a lot using the Start Lot or Perform Single Part Test button. Use this callback sequence to launch a dialog box to prompt an operator to enter lot information or to retrieve lot information programmatically, such as from a database. The default implementation of the `ConfigureLotSettings` callback sequence launches the default Configure Lot Settings dialog box.
- ConfigureLotWhenStartingLot option—Set this property on the SemiconductorModuleManager object using the TSM Application API to true when you want to customize an operator interface to automatically retrieve lot settings at the beginning of the execution of each lot. When you start the execution of a lot, TSM calls the `ConfigureLotSettings` callback sequence, which you can configure to retrieve lot information programmatically, such as from a database. The default setting for the `ConfigureLotWhenStartingLot` property is `False`.
- GetLotSettings callback—TSM calls the `GetLotSettings` callback sequence as execution begins. Use this callback if you want to programmatically override a lot setting that might have been set in the `ConfigureLotSettings` callback sequence and you want to always do so at run time. The default implementation of the `GetLotSettings` callback sequence sets the `LotSettings.Standard.JobName` and `LotSettings.Standard.JobRevision` values based on the value of `LotSettings.Standard.MainSequenceFilePath`.

Complete the following steps to override the default `ConfigureLotSettings` or `GetLotSettings` callback sequence and customize the behavior for obtaining lot settings.

1. Determine whether a sequence file named `SemiconductorModuleCallbacks.seq` exists in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule` directory. If the sequence file does not exist, create it and ensure that it does not contain any sequences.
2. Copy the ConfigureLotSettings or GetLotSettings callback sequences from the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates\SemiconductorModuleCallbacks.seq` file to the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` file and make changes to the copy.
3. To customize the default Configure Lot Settings dialog box in the ConfigureLotSettings callback sequence, complete the following additional steps.
    a. Copy the Display Configure Lot Settings Dialog step from the `ConfigureLotSettings` callback sequence of the `<TestStand>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` file to the `ConfigureLotSettings` callback sequence of the `<TestStand><Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` file.
    b. Copy the contents of `<TestStand>\Components\Callbacks\NI_SemiconductorModule\Source\LotSettingsDialogs` to `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\Source\LotSettingsDialogs` and make changes to the copy of the LabVIEW project. If you are making extensive customizations to the LabVIEW project, use the Debugging the Lot Settings dialog box to enable automatic testing when you makes changes to the lot settings.
    c. Rebuild the packed project library build specification in the project to update the copy of the LabVIEW packed project library.

Complete the following steps to change the value of the `ConfigureLotWhenStartingLot` property in the default operator interface.

1. Copy the contents of the `<TestStand>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp>` directory to the `<TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp>` directory.

   Note You must manually create the `<TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp>` directory if it does not already exist.

2. Modify the files in the `<TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp>` directory to find a reference to the SemiconductorModuleManager object and set the `ConfigureLotWhenStartingLot` property to `True`.

See Also

Customizing the Behavior for Obtaining Station Settings

TSM Application API

Debugging the Lot Settings Dialog Box (TSM)

If you customize the Lot Settings dialog box, you can use the following steps to automatically test new changes you make to the lot settings. With these settings configured, TestStand Semiconductor Module (TSM) calls the VIs directly when configuring the lot settings so any changes you make are reflected immediately. You do not have to rebuild the packed project library.

1. Complete the following steps to configure TSM to call the Lot Settings dialog box source VIs instead of a packed project library.
   a. In the TestStand Sequence Editor, select **Configure»Adapters** and select **LabVIEW**.
   b. Click **Configure** and select **LabVIEW Development System**.

c. Click **OK** in the LabVIEW Adapter Configuration dialog box.
d. Click **Done** to close the Adapter Configuration dialog box.
e. Open the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file.
f. In the Display Configure Lot Settings Dialog step of the ConfigureLotSettings sequence, click the **Module** tab.
g. Click **Advanced Settings** and remove the checkmark from the **Always Run VI in LabVIEW Run-Time Engine** checkbox.
h. Click **Close** to close the LabVIEW Advanced Settings dialog box.
i. On the **Module** tab, change the VI path from the default value of LotSettingsDialogs.lvlibp\Configure Lot Settings.vi to Configure Lot Settings.vi.
j. Save the sequence file.

2. Complete the following steps to configure the Lot Settings dialog box VIs for debugging.
   a. Open the Configure Lot Settings VI.
   b. Select **File»VI Properties** to open the VI Properties dialog box.
   c. In the **Category** drop-down menu, select **Window Appearance**.
   d. Click **Customize** to open the Customize Window Appearance dialog box. View and record the current custom settings so you can restore these settings when you finish debugging.
   e. Close the Customize Window Appearance dialog box.
   f. In the VI Properties dialog box, select **Default**.
   g. Click **OK** to close the VI Properties dialog box.
   h. Save the VI.
3. Debug the Lot Settings dialog box VIs.
4. When you finish debugging the VIs, complete the following steps to restore the settings to the previous state.

a. Open the Configure Lot Settings VI.
b. Select **File»VI Properties** to open the VI Properties dialog box.
c. In the **Category** drop-down menu, select **Window Appearance**.
d. Select **Custom** and then click **Customize** to launch the Customize Window Appearance dialog box.
e. Restore the custom settings you recorded in step 2d.
f. Click **OK** to close the Customize Window Appearance dialog box.
g. Click **OK** to close the VI Properties dialog box.
h. Save the VI.

5. Rebuild the packed project library specification as instructed in the Customizing the Behavior for Obtaining Lot Settings topic to use the modified version of the dialog box with the restored settings.
6. Complete the following steps to restore TSM to call a packed project library instead of the Lot Settings dialog box.
   a. In the Display Configure Lot Settings Dialog step of the `ConfigureLotSettings` sequence, click the **Module** tab.
   b. Click **Advanced** settings and select the **Always Run VI in LabVIEW Run-Time Engine** checkbox.
   c. On the **Module** tab, set the VI path to the `LotSettingsDialogs.lvlibp\Configure Lot Settings.vi`.
   d. Save the sequence file.

ConfigureLotSettings Callback (TSM)

The default implementation of the `ConfigureLotSettings` callback sequence launches the default Configure Lot Settings dialog box, in which the operator can configure the lot settings. You can override this callback sequence to customize the behavior. You can also use the ConfigureLotWhenStartingLot option on the Semiconductor Module Manager to customize an operator interface to automatically call the `ConfigureLotSettings` callback sequence at the beginning of the execution of each lot.

The `ConfigureLotSettings` callback sequence accepts the following parameters:

- **LotSettings** [In/Out]—An instance of the NI_SemiconductorModule_LotSettings data type. Configure the `ConfigureLotSettings` callback sequence to assign values for all the properties that you require.

  If you implement a dialog box for users to manually enter lot information, ensure that the dialog box obtains values for all the properties of the **LotSettings** parameter that you require. If the default properties of the **LotSettings** parameter do not meet your requirements, you can add properties to the NI_SemiconductorModule_CustomLotSettings data type. The properties you add to the data type appear in the `Custom` container of the **LotSettings** parameter.

  The `LotSettings.Standard.MainSequenceFilePath` property specifies the file path of the test program main sequence file to use with the current test lot.

  TSM records the lot setup time in the `LotSettings.Standard.SetupTime` property and saves lot settings to disk when the `ConfigureLotSettings` callback sequence sets the **Canceled** parameter to `False`. The value of the `LotSettings.Standard.SetupTime` property determines the value of the SETUP_T field in the Master Information Record (MIR) of version 4 of the Standard Test Data Format (STDF).

- **Canceled** [Out]—Configure the `ConfigureLotSettings` callback sequence to set this parameter to `True` if the operator cancels edits in the Configure Lot Settings dialog box. TSM saves the values of the **LotSettings** and **StationSettings** parameters to disk only when the **Canceled** parameter is `False`.

- **SemiconductorModuleManager** [In]—A reference to the Semiconductor Module Manager object that you use to call utility methods in the TSM Application API.

- **StationSettings** [In/Out]—An instance of the NI_SemiconductorModule_StationSettings data type. Use this parameter to modify station settings you require. Typically, the ConfigureStationSettings callback sequence sets station settings, but you might want to set certain station settings while configuring lot settings. TSM saves station settings to disk when the ConfigureLotSettings callback sequence sets the **Canceled** parameter to `False`.

See Also

Accessing Lot Settings from a Test Program

GetLotSettings Callback (TSM)

TSM calls the `GetLotSettings` callback sequence to programmatically obtain lot settings without requiring much, if any, operator interaction when execution begins. You can also use the ConfigureLotWhenStartingLot option on the Semiconductor Module Manager to customize an operator interface to automatically call the `ConfigureLotSettings` callback sequence at the beginning of the execution of each lot.

The default implementation of the `GetLotSettings` callback sequence sets the `LotSettings.Standard.JobName` and `LotSettings.Standard.JobRevision` values based on the value of `LotSettings.Standard.MainSequenceFilePath`. You can override this callback sequence to customize the behavior for when TSM attempts to determine lot settings values at run time. You can also use the ConfigureLotSettings callback sequence to prompt an operator to manually enter lot information in a dialog box or to use another mechanism that requires user input.

The `GetLotSettings` callback sequence accepts the following parameters:

- **LotSettings** [In/Out]—An instance of the NI_SemiconductorModule_LotSettings data type. Enter values for all the properties that you require.

  If you implement a mechanism for programmatically obtaining lot information, ensure that the mechanism obtains values for all the properties

of the **LotSettings** parameter that you require. If the default properties of the **LotSettings** parameter do not meet your requirements, you can add properties to the NI_SemiconductorModule_CustomLotSettings data type. The properties you add to the data type appear in the `Custom` container of the **LotSettings** parameter.

TSM records the lot setup time in the `LotSettings.Standard.SetupTime` property when the `GetLotSettings` callback sequence sets the **UpdateSetupTime** parameter to `True`. When you customize the `GetLotSettings` callback sequence, set the **UpdateSetupTime** parameter to `True` when the callback sequence updates the **LotSettings** parameter.

- **UpdateSetupTime** [Out]—Configure the `GetLotSettings` callback sequence to set this parameter to `True` when the callback sequence modifies the values of the **LotSettings** parameter. If the callback sequence does not modify the values of the LotSettings parameter, configure the callback sequence to set this parameter to `False`.

TSM records the lot setup time in the `LotSettings.Standard.SetupTime` property when the `GetLotSettings` callback sequence sets this parameter to `True`. The value of the `LotSettings.Standard.SetupTime` property determines the value of the SETUP_T field in the Master Information Record (MIR) of version 4 of the Standard Test Data Format (STDF).

See Also

Accessing Lot Settings from a Test Program

Accessing Lot Settings from a Test Program (TSM)

Use the Get Test Information step to access standard and custom lot settings in a test program sequence file.

Standard Lot Settings

The following table lists the properties of the NI_SemiconductorModule_StandardLotSettings data type, the fields of the Master Information Record (MIR) of the Standard Test Data Format (STDF) version 4 specification that the STDF Log result processing plug-in sets using the properties, or other purpose of each property.

Note Use the Types window in the TestStand Sequence Editor to review properties on the standard type or to add properties to the custom type.

Property Name	Corresponding STDF Record Field or Other Purpose
ActiveConfigurationName	Specifies the name of the test program configuration to use to obtain standard lot settings and custom test conditions.
AuxiliaryDataFile	MIR: AUX_FILE
BurnInTime	MIR: BURN_TIM
CommandModeCode	MIR: CMOD_COD
CorrelationOffsetsFileAbsolutePath	Contains the absolute path of the correlation offsets file the Load Correlation Offsets step loads. Do not set this property. TSM sets this property at run time. If the test program does not use correlation offsets, this property is an empty string. If the test program executes multiple Load Correlation Offsets steps, this property contains the path loaded by the step that executed last.
DateCode	MIR: DATE_COD
DeviceDesignRevision	MIR: DSGN_REV
EngineeringLotId	MIR: ENG_ID
FabricationProcessId	MIR: PROC_ID
JobName	MIR: JOB_NAM (Default value is the test program sequence filename without the file extension.)
JobRevision	MIR: JOB_REV (Default value is the test program sequence file version.)

LimitsFileAbsolutePath	Contains the absolute path of the test limits file that TSM used to import test limits at run time. Do not set this property. TSM sets this property at run time.
LimitsFileRelativePath	Specifies the test limits file to use for importing test limits at run time. Use the Test Program Editor to specify the limits file for a test program configuration. TSM sets this property at run time to the limits file the active test program configuration specifies.
LimitsFileReplaceTests	Specifies whether TSM deletes existing tests before importing limits from a test limits file that `LimitsFileRelativePath` specifies. A value of `True` corresponds to the **Replace all tests in matching steps** import mode. A value of `False` corresponds to the **Update limits in matching tests** import mode.
LotId	MIR: LOT_ID
LotSize	Corresponds to the **Estimated Lot Size** option in the Configure Lot Settings dialog box. An inline QA algorithm can use this property to determine for which DUTs to enable inline QA.
MainSequenceFilePath	Specifies the file path of the test program main sequence file to use with the current test lot.
MIRUserText	MIR: USER_TXT
OperationFrequency	MIR: OPER_FRQ
OperatorName	MIR: OPER_NAM
PackageType	MIR: PKG_TYP
PartType	MIR: PART_TYP
ProductFamilyId	MIR: FAMLY_ID
ProtectionCode	MIR: PROT_COD
RequireEveryStepToBeInLimitsFile	Specifies whether TSM requires that every step or test in the test program sequence file have a corresponding entry in the test limits file when importing limits from a test limits file.
RetestCode RetestCount*	MIR: RTST_COD

ROMCodeId	MIR: ROM_COD
SetupTime	MIR: SETUP_T This value represents the setup time in seconds since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC). UTC is also known as Greenwich mean time. The STDF Log result processing plug-in converts this value to local time when storing the value in the SETUP_T field.
SublotId	MIR: SBLOT_ID TSM sets the SBLOT_ID field to the wafer ID obtained from the prober driver when you enable the Generate One File per Wafer STDF option and the SublotId lot setting property value is empty.
SupervisorName	MIR: SUPR_NAM
TestCode	MIR: TEST_COD
TestFlowId	MIR: FLOW_ID
TestModeCode	MIR: MODE_COD
TestSetupId	MIR: SETUP_ID
TestSpecificationName	MIR: SPEC_NAM
TestSpecificationVersion	MIR: SPEC_VER
TestTemperature	MIR: TST_TEMP
UseEmbeddedLimitsFileData	Indicates whether the limits file data is embedded in the test program sequence file. Do not set this property. TSM sets this property at run time based on the active configuration.
Wafer.WaferSize	WCR: WAFR_SZ
Wafer.DieHeight	WCR: DIE_HT
Wafer.DieWidth	WCR: DIE_WID
Wafer.Units	WCR: WF_UNITS
Wafer.FlatOrientation	WCR: WF_FLAT
Wafer.CenterXCoordinate	WCR: CENTER_X
Wafer.CenterYCoordinate	WCR: CENTER_Y
Wafer.PositiveXDirection	WCR: POS_X
Wafer.PositiveYDirection	WCR: POS_Y
*	

```
If (RetestCode is not empty string) then
RTST_COD = RetestCode
Else If (RetestCount is in the range [0-9]) then
RTST_COD = '0' - '9'
Else
RTST_COD = space
```

NI_SemiconductorModule_LotSettings Data Type

The NI_SemiconductorModule_LotSettings data type defines the properties for each instance of the LotSettings sequence parameter, which the test program can access.

TSM obtains the values for the NI_SemiconductorModule_LotSettings data type from the ConfigureLotSettings and GetLotSettings callback sequences. The NI_SemiconductorModule_LotSettings data type contains the following properties:

- **Standard**—An instance of the NI_SemiconductorModule_StandardLotSettings data type. This property contains the standard lot settings TSM recognizes.
- **Custom**—An instance of the NI_SemiconductorModule_CustomLotSettings data type. By default, this property is an empty container. You can add properties to the NI_SemiconductorModule_CustomLotSettings data type to add custom lot settings that appear in this container.
- **CustomTestConditions**—An empty container. TSM adds properties to this container at run time that match the custom test conditions in the active test program configuration.

See Also

GetLotSettings Callback

Accessing Lot Settings from a Test Program

Get Test Information Step

Reports and Data Logs (TSM)

You can generate the following types of TSM reports and data logs. Enable and configure the corresponding TSM result processing plug-in to generate the report or data log.

- Standard Test Data Format (STDF) Log—Standard file format for storing test station information, lot information, and semiconductor test result data. Use the STDF Log to generate an STDF log file that complies with the STDF version 4 specification. Use the STDF Log Options dialog box to specify settings for the STDF log.
- Lot Summary Report—Text file that contains a summary of the semiconductor test results for the current lot of DUTs (test lot). Use the Lot Summary Report to gather lot, site, software bin, and hardware bin results in a human-readable ASCII text format. Use the Lot Summary Options dialog box to specify settings for the Lot Summary Report.
- Debug Test Results Log—Human-readable text file that contains the measurement values and test limits for each test that executes on each site. The Debug Test Results Log result processor generates separate files for each site in the test program. Use the Debug Test Results Log to debug measurements and test configurations as a lot executes. Use the Debug Test Results Log Options dialog box to specify settings for the Debug Test Results Log.

Note Enabling the Debug Test Results Log might affect the performance of a test program.

- CSV Test Results Log—Human-readable text file that contains data in a comma-separated values text file, which provides better performance than the Debug Test Results Log result processor in a production environment. The CSV Test Results Log result processor generates a single file for all sites in the test program. You can open the `.csv` file directly in a spreadsheet application for analysis or to correlate test results. Use the CSV Test Results Log Options dialog box to specify settings for the CSV Test Results Log.

Notes

- Keeping the Report pane open during execution negatively affects performance.
- The STDF Log, the Debug Test Results Log, and the CSV Test Results Log contain only semiconductor test result data from Semiconductor Multi Test steps. These data logs do not contain result data from other types of steps. If you use a Sequence Call step to call a sequence that contains Semiconductor Multi Test steps, the data logs **do** contain the test result data from those steps unless you use the Use New Thread sequence call option or the Use New Execution sequence call option. When using these sequence call options to execute a sequence asynchronously, you must use a Wait step to wait for the thread or execution for the data logs to contain the test result data from the sequence.
- You can configure the Lot Summary Report and Debug Test Results Logs in the Result Processing dialog box to display in the Report pane in the sequence editor and in the reports dialog box in the default operator interface. You cannot view the STDF Log or CSV Test Results Log in the sequence editor or default operator interface.

You can disable result recording by selecting the **Disable Result Recording for All Sequences** option on the Execution tab of the Station Options dialog box.

See Also

GetReportFileName Callback

Lot Summary Options Dialog Box

Specifying Report and Data Log Filenames

STDF Log Options Dialog Box

Debug Test Results Log Options Dialog Box

CSV Test Results Log Options Dialog Box

Enabling and Configuring TSM Result Processing Plug-ins

Complete the following steps to enable and configure one or more TSM result processing plug-ins to generate TSM reports and data logs for the current lot of DUTs (test lot).

1. In the TestStand Sequence Editor, select **Configure»Result Processing** to launch the Result Processing dialog box.
2. If the Output Name column already contains an item for the type of report or data log you want to generate, skip to step 5 to configure the result processing plug-in.
3. Enable the **Show More Options** control to display additional options to insert or delete instances of result processing plug-ins.
4. Click the **Insert New** > button and select the item from the list of result processing plug-ins that corresponds to the type of report or data log you want to generate. The result processing plug-in you selected now appears in the Output Name column.
5. Ensure that the Enabled column contains a checkmark for the type of report or data log that you want to generate.
6. Click the **Options** button for the result processing plug-in to launch the related result processing plug-in Options dialog box.
7. In the Destination Directory option, specify the absolute path of the directory in which you want TSM to create the report or data log file. Leave the control blank if you want TSM to create the report or data log file in the same directory as the test program main sequence file.

Note TSM result processing plug-ins use the combination of the destination directory you specify in the Options dialog box and the report or data log filename the GetReportFileName callback sequence

returns to determine the absolute path of the report or data log file. You can modify a copy of the GetReportFileName sequence to customize the destination directory and filename of the report or data log file.

8. Click **OK** to close the result processing plug-in Options dialog box and click **OK** again to close the Result Processing dialog box.

Specifying Report and Data Log Filenames (TSM)

TSM result processing plug-ins determine the name of the report or data log file by calling the GetReportFileName callback sequence once for each active result processing plug-in instance. The `GetReportFileName` callback sequence is located in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` or `<TestStand>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` file.

TSM result processing plug-ins use the combination of the destination directory you specify in the corresponding result processing plug-in Options dialog box and the report or data log filename the `GetReportFileName` callback sequence returns to determine the absolute path of the report or data log file. When you customize the log filename for the Debug Test Results Log, use unique filenames for each test site. The Debug Test Results Log result processing plug-in appends results from each DUT to the corresponding site log.

You can customize the filenames and locations in which TSM result processing plug-ins create report and data log files by modifying the `GetReportFileName` callback. The `GetReportFileName` callback sequence contains parameters that specify result processing plug-in information, lot information, station information, date and time information, and a destination directory. You can use the TSM Application API in a code module the `GetReportFilename` callback sequence calls to access batch and site run-time data, including wafer information.

Complete the following steps to customize TSM report and data log filenames.

1. Determine whether a sequence file named `SemiconductorModuleCallbacks.seq` exists in the `<TestStand Public>\Components\Callbac`

ks\NI_SemiconductorModule directory. If the sequence file does not exist, create it and ensure that it does not contain any sequences.

2. Copy the GetReportFileName callback sequence from the <TestStand>\Components\Modules\NI_SemiconductorModule\Templates\SemiconductorModuleCallbacks.seq file to the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file and make changes to the copy.

   You can use the **ModelPlugin**, **ModelThreadType**, **ModelData**, **LotSettings**, **StationSettings**, **StartDate**, and **StartTime** sequence parameters, as necessary, to assign a value to the **ReportFileName** sequence parameter and optionally to the **ReportDestinationDirectory** sequence parameter.

GetReportFileName Callback (TSM)

The STDF Log, Lot Summary Report, CSV Test Results Log, and Debug Test Results Log result processing plug-ins determine the name of the report or data log file by calling the GetReportFileName callback sequence once for each active result processing plug-in instance. The STDF Log, CSV Test Results Log, and Lot Summary Report result processing plug-ins call the callback once per wafer when you enable the Generate One File per Wafer option. The GetReportFileName callback sequence is located in the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq or the <TestStand>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file.
The GetReportFileName callback sequence accepts the following parameters:

- **ModelPlugin** [In] —An instance of the NI_ModelPlugin data type. Use the Parameters.ModelPlugin.Base.SequenceFileName property to determine which result processing plug-in called the callback sequence. Possible values include the following:

  - STDF Log: NI_SemiconductorModule_StdfGenerator.seq

- Lot Summary Report: `NI_SemiconductorModule_LotSummaryReportGenerator.seq`
- Debug Test Results Log: `NI_SemiconductorModule_TestResultsLogGenerator.seq`
- CSV Test Results Log: `NI_SemiconductorModule_CSVTestResultsLogGenerator.seq`

- **ModelThreadType** [In] —An instance of the NI_ModelThreadType data type. Use values in this container to determine whether the filename corresponds to a specific site or to all sites. For example, the STDF Log result processing plug-in creates a single file for all sites, and the Debug Test Results Log result processing plug-in calls the `GetReportFileName` callback once for each site to create a separate file for each site.
- **ModelData** [In] —Contains information about the process model used to test the current lot.
- **LotSettings** [In] —An instance of the NI_SemiconductorModule_LotSettings data type. Use values in this container to append lot settings, such as `Standard.JobName` and `Standard.LotId`, to the filename. The TSM result processing plug-ins call the `GetReportFileName` callback sequence after the ConfigureLotSettings and GetLotSettings callback sequences, which might have set values in this container.
- **StationSettings** [In] —An instance of the NI_SemiconductorModule_StationSettings data type. Use values in this container to append station settings, such as `Standard.NodeName` and `Standard.TestFacilityId`, to the filename. TSM result processing plug-ins call the `GetReportFileName` callback sequence after the ConfigureStationSettings and GetStationSettings callback sequences, which might have set values in this container.
- **StartDate** [In]—An instance of the DateDetails data type that represents the date that testing began. For the STDF Log and Lot Summary Report, this parameter contains the date that the current wafer started testing if the Generate One File per Wafer option is enabled. Use values in this container to append the lot test start date to the filename.

- **StartTime** [In]—An instance of the TimeDetails data type that represents the time that testing began. For the STDF Log and Lot Summary Report, this parameter contains the time that the current wafer started testing if the Generate One File per Wafer option is enabled. Use values in this container to append the lot test start time to the filename.
- **ReportDestinationDirectory** [In/Out]—Contains the report or data log file destination directory, which you configured in the related result processing plug-in Options dialog box. Change the value of this parameter to store the report or data log file in a different location.
- **ReportFileName** [Out]—The report or data log filename. TSM concatenates the values of the **ReportDestinationDirectory** and **ReportFileName** parameters to generate the report or data log file absolute path.

In addition to using the parameters to generate a report filename, you can use the TSM Application API in a code module the `GetReportFilename` callback sequence calls to access batch and site run-time data, including wafer information.

Standard Test Data Format (STDF) Log (TSM)

The Standard Test Data Format (STDF) Log is a standard file format for storing test station information, lot information, and semiconductor test result data.

Notes

- When you retest a DUT, the STDF file includes the PIR, PRR, PTR, and FTR records for every test run, including retests. The WRR, PCR, HBR, and SBR summary records include only the results of the last retest and do not include retested results. The TSR summary record includes the counts of all tests run, including retests.
- All string fields in the STDF log must use only ASCII characters. If lot settings, such as `TestTemperature`, and station settings, such as `LoadBoardId`, use

non-ASCII characters, TSM replaces those characters with the question mark character (?) when writing those values to the STDF log.

- You cannot view the STDF Log in the sequence editor or default operator interface.

Logging Custom Data to the STDF Log File (TSM)

The STDF Log result processing plug-in automatically logs data to the STDF log file about the tester configuration and test execution. You can customize tester, part, wafer, and text data to log in the STDF log file.

Data Related to Tester Information

The Master Information Record (MIR), Site Description Record (SDR), and Wafer Configuration Record (WCR) in the STDF log file store information about the configuration of the tester itself. TSM automatically logs data from the NI_SemiconductorModule_StandardLotSettings and NI_SemiconductorModule_StandardStationSettings data types to these records. You can customize the data stored in the STDF log records by using callbacks to modify the data in the containers within TestStand.

TSM stores the following values in the MIR in addition to values TSM obtains from the NI_SemiconductorModule_LotSettings and NI_SemiconductorModule_StationSettings data types:

MIR Field Name	Value
EXEC_TYPE	NI STS Software **Note** You can customize this field.
EXEC_VER	If STS Software is installed: <STS Software Version> Otherwise: <TSM year-based version (<TSM full version>)> (For example, 2016 (16.0.0.49152))

Data Related to Part Information

The Part Results Record (PRR) in the STDF log file stores information about each tested part, including the Part ID, X and Y coordinates of the location of the part on the die, and a text description for the part. You can customize the data TSM logs from the UUT container to these fields within the PRR record.

Data Related to Wafers

The Wafer Information Record (WIR) and Wafer Result Record (WRR) in the STDF log file store information related to wafer tests. If you are performing wafer testing, you can customize the data stored in these records by modifying the value of the **WaferRuntimeData** parameter in the StartOfTest handler/prober driver entry point.

Text Data

If the data you want to add to the STDF log file is not included in any of the above records, you can include text data as a Datalog Text Record (DTR) in the STDF log file by using the following techniques:

- **Adding Text Data for a Part Using Additional Results**—You can use the Additional Results panel of the Step Settings pane to add a DTR record after all the part test records associated with the step. To configure the Additional Results panel to generate a DTR record, set the **Name** option of the result to `NI.STDF.DTR` and set the **Value to Log** option to the value you want to store in the DTR record. You can create multiple results with the name `NI.STDF.DTR`, and each result appears in the STDF log file as a separate DTR record.
- **Adding Text Data between Parts**—If the text data you want to log is not associated with a specific part, you can insert DTR records between parts using a .NET Action step in a callback sequence or in a custom result processor model plug-in.

Logging Failed Cycle Information from NI-Digital Pattern Driver to STDF Log File (TSM)

You can fetch and publish failed cycle information from the NI-Digital Pattern Driver and log the information in the STDF Log file.

Note TSM does not support logging failed cycles when using multiple NI-Digital Pattern Driver sessions to burst a pattern for a site. If you need multiple instruments to burst a pattern for a site, combine those instruments into a group in the pin map so the test program uses a single session to burst the pattern.

Fetching and Publishing Failed Cycle Information in Code Modules (TSM)

Fetch and publish failed cycle information from an NI-Digital Pattern Driver with your LabVIEW or .NET application.

▾ LabVIEW

Complete the following steps to log failed cycle information from an NI-Digital Pattern driver.

1. Configure the NI-Digital Pattern Driver to acquire failed cycle information.
2. Fetch failed cycle information from the NI-Digital Pattern Driver after bursting a pattern.
3. Publish the fetched cycle information to TSM.

Configure the NI-Digital Pattern Driver to Acquire Failed Cycle Information

Use the following NI-Digital Pattern Driver API in code modules to configure your NI-Digital Pattern instruments to acquire failed cycle information and log this information in the TSM STDF Log file.

Place an niDigital property node on the block diagram and set the following properties:

- **Triggers»History RAM»Type**—Set to **First Failure** to start acquiring information on the first failed cycle.

- **History RAM»Cycles to Acquire**—Set to **Failed Cycles** to acquire only failed cycles.
- **History RAM»Max Samples to Acquire per Site**—Set to the maximum number of failed cycles you want to acquire per site.

Fetch Failed Cycle Information from the NI-Digital Pattern Driver

After configuring the NI-Digital Pattern instrument and bursting a pattern, use the niDigital Fetch Multi-Site History RAM Information VI to fetch the failed cycle information and organize the information into a format the TSM Code Module API can consume.

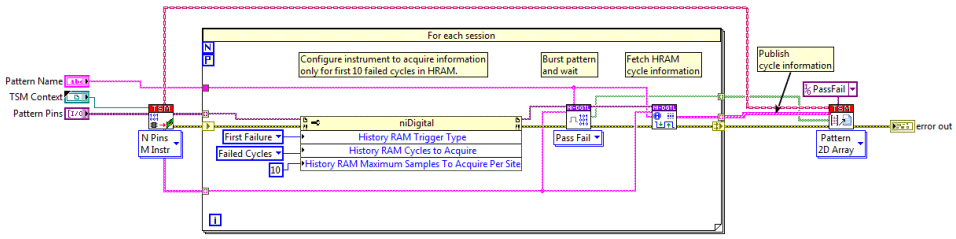
Note The niDigital Fetch Multi-Site History RAM Information VI does not appear on the NI-Digital Pattern Driver palette. Browse to and select the VI from the NI-Digital Pattern Driver library file (`niDigital.llb`), located in the `<LabVIEW>\instr.lib\niDigital` directory.

Publish Failed Cycle Information to TSM

Wire the output of the niDigital Fetch Multi-Site History RAM Information VI to the **History RAM Information** input of the Publish Pattern Results VI in the TSM Code Module API. TSM logs the failed cycle information in Functional Test Records (FTRs) in the STDF Log file.

Code Module Example

The following figure shows how to fetch and publish failed cycle information in a code module. This example logs a maximum of 10 failed cycles.



▼.NET

Complete the following steps to log failed cycle information from an NI-Digital Pattern driver.

1. Configure the NI-Digital Pattern Driver to acquire failed cycle information.
2. Fetch failed cycle information from the NI-Digital Pattern Driver after bursting a pattern.
3. Publish the fetched cycle information to TSM.

Configure the NI-Digital Pattern Driver to Acquire Failed Cycle Information

Use the following NI-Digital Pattern Driver API in code modules to configure your NI-Digital Pattern instruments to acquire failed cycle information and log this information in the TSM STDF Log file.

```
// Configure the instrument to start acquiring information on the first failed cycle
session.Trigger.HistoryRamTrigger.TriggerType =
HistoryRamTriggerType.FirstFailure;

// Configure the instrument to acquire only failed cycles
session.HistoryRam.CyclesToAcquire = HistoryRamCycle.Failed;

// Set the maximum number of failed cycles you want to acquire per site
session.HistoryRam.MaximumSamplesToAcquirePerSite = 10;
```

Fetch Failed Cycle Information from the NI-Digital Pattern Driver

After configuring the NI-Digital Pattern instrument and bursting a pattern, use the `FetchMultisiteHistoryRamInformation` extension method to fetch the failed cycle information and organize the information into a format the TSM Code Module API can consume.

Note The TSM .NET cde module API provides the `FetchMultisiteHistoryRamInformation` extension method for the NIDigital .NET class. It is not part of the NI-Digital Pattern Driver API.

Publish Failed Cycle Information to TSM

Pass the return value from the `FetchMultisiteHistoryRamInformation` method to the `historyPamCycleInformation` parameter of the `PublishPatternResults` method on the pin query context class in the TSM .NET code module API. TSM logs the failed cycle information in Functional Test Records (FTRs) in the STDF Log file.

Code Module Example

The following example shows how to fetch and publish failed cycle information in a code module. This example code module logs a maximum of 10 failed cycles.

```
public static void PerformTest(ISemiconductorModuleContext
semiconductorModuleContext, string patternName, string[] patternPins)
{

var pinQueryContext =
semiconductorModuleContext.GetNIDigitalPatternSessionsForPattern(patternPins,
out var sessions, out var siteLists);
var passFailResultsPerSession = new bool[sessions.Length][];
var historyRamCycleInformationPerSession = new
NIDigitalHistoryRamCycleInformation[sessions.Length];

Parallel.For(0, sessions.Length, i =>
{

var session = sessions[i];
var siteList = siteLists[i];

// Configure instrument to acquire cycle information only for first 10 failed cycles
session.Trigger.HistoryRamTrigger.TriggerType =
HistoryRamTriggerType.FirstFailure;
session.HistoryRam.MaximumSamplesToAcquirePerSite = 10;
session.HistoryRam.CyclesToAcquire = HistoryRamCycle.Failed;

// Burst pattern
passFailResultsPerSession[i] = session.PatternControl.BurstPattern(patternName,
```

```
patternName, true, new TimeSpan(0, 0, 2));

// Fetch history RAM cycle information
historyRamCycleInformationPerSession[i] =
session.FetchMultisiteHistoryRamInformation(siteList, patternName);
});

pinQueryContext.PublishPatternResults(passFailResultsPerSession,
historyRamCycleInformationPerSession, "PassFail");
}
```

See Also

Grouping Instruments

Failed Cycle Information in Functional Test Records (FTRs) (TSM)

When you publish failed cycle information for a test a Semiconductor Multi Test step defines, TSM stores the failed cycle information in the Functional Test Record (FTR) for that test. Because an FTR can contain failed cycle information only for a single cycle, TSM writes multiple FTRs to the STDF Log file if the data contains information for multiple failed cycles.

TSM stores the failed cycle information in the following fields of the FTR:

Field	Value
CYCL_CNT	Cycle number from the NI-Digital Pattern Driver History RAM information. If this value is larger th an a 32-bit number, TSM does not set this field.
REL_VADR	Vector number from the NI-Digital Pattern Driver History RAM cycle information.
NUM_FAIL	Number of pins in the pattern with 1 or more fail ures.

RTN_ICNT	Number of actual pin states from the NI-Digital Pattern Driver History RAM cycle information.
PGM_ICNT	Number of expected pin states from the NI-Digital Pattern Driver History RAM cycle information.
RTN_INDX	Array of pin indexes that corresponds to the actual pin states from the NI-Digital Pattern Driver History RAM cycle information. Each pin index refers to a pin index defined in a Pin Map Record (PMR) for the pin.
RTN_STAT	Actual pin states from the NI-Digital Pattern Driver History RAM cycle information.
PGM_INDX	Array of pin indexes that corresponds to the expected pin states from the NI-Digital Pattern Driver History RAM cycle information. Each pin index refers to a pin index defined in a PMR for the pin.
PGM_STAT	Expected pin states from the NI-Digital Pattern Driver History RAM cycle information.
FAIL_PIN	Bitfield that corresponds to the values in the per-pin pass fail array from the NI-Digital Pattern Driver History RAM cycle information. Each bit corresponds to a pin index defined in a PMR record. For example, if the bit in position 1 is set, the pin defined in the PMR with a pin index of 1 failed for the cycle.
VECT_NAM	Pattern name from the NI-Digital Pattern Driver History RAM cycle information.
TIME_SET	Time set name from the NI-Digital Pattern Driver History RAM cycle information.

Expected Pin State Information in STDF Log Files

The failed cycle information includes the expected pin states from the NI-Digital Pattern Driver History RAM cycle information. TSM stores this information in the PGM_STAT field of the FTR. The pin states from the NI-Digital Pattern Driver History RAM cycle information correspond to the pin states defined in the STDF V4 specification according to the following table.

NI-Digital Pattern Driver Pin State	STDF Pin State Value	STDF Pin State Description

0	0	Drive Low
1	1	Drive High
L	2	Expect Low
H	3	Expect High
M	4	Expect Midband
V	5	Expect Valid (not midband)
X	6	Do not drive or compare

Actual Pin State Information in STDF Log Files

The failed cycle information includes the actual pin states from the NI-Digital Pattern Driver History RAM cycle information. TSM stores this information in the RTN_STAT field of the FTR. The pin states from the NI-Digital Pattern Driver History RAM cycle information correspond to the pin states defined in the STDF V4 specification according to the following table.

NI-Digital Pattern Driver Pin State	STDF Pin State Value	STDF Pin State Description
L	0	0 or Low
H	1	1 or High
M	2	Midband
V*	11	—

* This pin state is not defined in the STDF specification. The NI-Digital Pattern Driver uses this state only when $V_{ol} > V_{oh}$ and the pin voltage level is between V_{ol} and V_{oh}.

Edge Multiplier Representation

When a pin uses an edge multiplier of 2, the NI-Digital Pattern Driver includes two pin states for that pin in the History RAM cycle information. For these pins, the FTR contains duplicate pin indexes to represent both pin states in the PGM_STAT and RTN_STAT fields.

For example, if the pattern uses two pins with PMR pin indexes 1 and 2, and pin index 1 uses an edge multiplier of 2, but pin index 2 uses an edge multiplier of 1, the pin index and pin state fields in the FTR appear as shown in the following table.

Field	Value

PGM_INDX	[1, 1, 2]
PGM_STAT	[L, L, H]
RTN_INDX	[1, 1, 2]
RTN_STAT	[0, 0, 1]

Modifying Number of Results to Include in STDF Log Files (TSM)

By default, the STDF log file includes all test results. You can change the default behavior.

Complete the following steps to limit the number of results to include in the STDF log file.

1. Launch the STDF Log Options dialog box.
2. Enable the **Limit Number of Test Data Records** option.
3. Use the **Log test data for only one out of every** option to specify the number of tests to include in the STDF log file.

Customizing STDF MIR, SDR, and WCR Field Values (TSM)

The STDF Log result processing plug-in sets the fields in the Master Information Record (MIR), Site Description Record (SDR), and Wafer Configuration Record (WCR) of the STDF version 4 specification by using the values of the properties in the `Standard` containers of the NI_SemiconductorModule_LotSettings and NI_SemiconductorModule_StationSettings data types. The `LotSettings.Standard.Wafer` container contains the property values that determine the WCR field values. TSM uses the following callback sequences, located in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` or `<TestStand>\Components\Module\NI_SemiconductorModule\Templates\SemiconductorModuleCallbacks.seq` file, to obtain the values of these data types:

- ConfigureLotSettings
- GetLotSettings
- ConfigureStationSettings
- GetStationSettings

You can create custom versions of these callback sequences to modify the values TSM obtains for the NI_SemiconductorModule_LotSettings and NI_SemiconductorModule_StationSettings data types, and therefore to modify the values the STDF Log result processing plug-in sets in the MIR, SDR, and WCR fields.

When using a handler/prober driver, TSM calls the handler/prober driver Setup entry point sequence after calling the GetLotSettings and GetStationSettings callbacks. The Setup entry point sequence can modify the values of the properties in the **LotSettings** and **StationSettings** parameters, which determine the values used in the MIR, SDR, and WCR fields. Typically, a prober driver sets the properties for the WCR fields by setting the property values in the `LotSettings.Standard.Wafer` container.

Customizing the MIR `EXEC_TYPE` Value

Complete the following steps to customize the MIR `EXEC_TYPE` value:

1. Using a text editor, open the `StdfGenerator.ini.example` file located in the `<TestStand Application Data>\Cfg\NI_Semiconductor Module` directory.
2. Modify the `StdfGenerator.ini.example` file to specify the MIR `EXEC_TYPE` value you want.

MIR `EXEC_TYPE` Value	Steps
Custom	a. Update the ExecType key with the `EXEC_TYPE` value you want. For example, ExecType = "iniExecType". b. Verify that the UseTSMVersionBasedExecTypeFromRegistry key is set to `false`.
Based on registry key	Set the UseTSMVersionBasedExecTypeFromRegistry key to `true`. If STS Software is installed, the MIR `EXEC_TYPE` value will be updated to `NI STS Development Software`. If STS Software is not installed, the MIR `EXEC_TYPE` value will

	be updated to `TestStand <year-based version> Semiconductor Module Runtime` (32 bit or 64 bit) based on the TestStand version specified in the registry key.

3. Go to **File>>Save as** and rename the modifed file to `StdfGenerator.ini`.

See Also

Customizing the Behavior for Obtaining Lot Settings

Customizing the Behavior for Obtaining Station Settings

Customizing STDF PRR Field Values (TSM)

The STDF Log result processing plug-in sets the PART_ID, PART_TXT, X_COORD, and Y_COORD fields in the Part Results Record (PRR) of the STDF version 4 specification by using the values of properties on the UUT data type as described in the following table. The handler/prober driver StartOfTest entry point can set any of these fields by setting the values of the output parameters to the StartOfTest entry point.

STDF PRR Field	UUT Property	Handler/Prober StartOfTest Output Parameter
PART_ID	SerialNumber	SitePartIds **Note** By default, if the handler/prober driver does not set the values in the **SitePartIds** parameter, TSM automatically assigns sequential numeric values to the `SerialNumber`

		property, which results in unique PART_ID field values. To disable the default behavior, set the `GenerateUniquePartIds` property of the NI_SemiconductorModule_StationSettings data type to `False` in the ConfigureStationSettings or GetStationSettings callback sequence. When the `GenerateUniquePartIds` property is `True`, TSM reassigns the same unique Part ID to the part when it is retested. Customize the behavior of `GenerateUniquePartIds` to assign a new unique Part ID to a part when it is retested.
PART_TXT	AdditionalData.NI.SemiconductorModule.PartText	SitePartTexts
X_COORD, Y_COORD	AdditionalData.NI.SemiconductorModule.DieCoordinates	SiteDieCoordinates

You can modify the PART_ID field value in the test program by setting the `SerialNumber` property value on the UUT object. You can modify the PART_TXT field value in the test program by setting the `AdditionalData.NI.SemiconductorModule.PartText` property value on the UUT object. You can access the UUT object using a parameter in the `PreMainSequence` or `PreBatch` callback sequence or by using the expression `RunState.Root.Locals.UUT` in a Statement step.

Customizing STDF WIR and WRR Field Values (TSM)

The STDF Log result processing plug-in sets the fields in the Wafer Information Record (WIR) and Wafer Result Record (WRR) of the STDF log file by using the values returned in the **WaferRuntimeData** parameter of the StartOfTest handler/prober driver entry point as shown in the following table.

STDF WIR or WRR Field	Handler/Prober StartOfTest Output Parameter
WAFER_ID	WaferRuntimeData.Identity.WaferId **Note** TSM automatically generates unique wafer IDs by setting the **WaferRuntimeData.Identity.WaferId** parameter value before calling the StartOfTest entry point. A prober driver can override the wafer IDs by modifying the parameter value.
FABWF_ID	WaferRuntimeData.Identity.FabWaferId
FRAME_ID	WaferRuntimeData.Identity.FrameId
MASK_ID	WaferRuntimeData.Identity.MaskId
USR_DESC	WaferRuntimeData.Identity.UserDescription
EXC_DESC	WaferRuntimeData.Identity.ExecDescription

Adding DTRs to the STDF Log File (TSM)

Using the TSM STDF result processor, you can insert Datalog Text Records (DTR) in various locations in the STDF log file. The method for creating DTRs differs depending on the following conditions and locations at which you insert the DTR.

Inserting a DTR after a Test Record

Use the Additional Results edit tab to add an additional result to the step with the name "NI.STDF.DTR" to insert a DTR after all Parametric Test Records (PTR) or Functional Test Records (FTR) associated with the step. Set the Value to Log option for the additional result to the value to store in the TEXT_DAT field of the DTR. The STDF Log result processor inserts a DTR into the STDF log file for each additional result with the name NI.STDF.DTR on steps in the main sequence. If the text in the Value to Log option is longer than 255 characters, the STDF Log result processor splits the text into character groups that are 255 characters or less and inserts a DTR for each group.

Note Additional results on steps in ProcessSetup, ProcessCleanup, and OnSiteTestingComplete are ignored. Refer to the Inserting a DTR between Parts section for information about inserting a DTR from a callback sequence in the STDF log file.

Inserting a DTR between Parts

You must call a method on the StdfResultProcessor class defined in the TSM assembly (NationalInstruments.TestStand.SemiconductorModule) to insert a DTR at locations other than between test records. Complete the following steps to call the CreateDtr method to insert a DTR in the STDF log file.

1. Create a .NET Action step in the sequence.
2. Complete the following steps on the Module tab:
    a. Set the **Assembly** option to NationalInstruments.TestStand.SemiconductorModule.dll.

b. Set the **Root Class** option to `NationalInstruments.TestStand.SemiconductorModule.StdfResultProcessor`.

c. Set the **.NET Invocation** option to `StdfResultProcessorSingleton.CreateDtr(System.String)`.

d. In the Parameters Table, use the Value column for the text parameter to insert the text to add to the TEXT_DAT field of the DTR. If the text in the Value column is longer than 255 characters, the STDF Log result processor splits the text into character groups that are 255 characters or less and inserts a DTR for each group.

You can add DTRs using callback sequences in the test program sequence file or by creating a custom result processor model plug-in. The actual implementation differs depending on whether you are performing wafer testing and whether you enable the Generate One File per Wafer option. In both cases, the custom result processor must not attempt to create DTRs if the STDF Log result processor is disabled. Confirm whether the STDF Log result processor is disabled at run-time and disable the custom result processor in the Model Plugin - Initialize entry point.

Adding DTRs for Non-Wafer Testing or When the Generate One Filer per Wafer Option is Disabled

The custom result processor that you create must appear after the STDF Log result processor in the order of result processors in the Result Processing dialog box. The following table lists the sequences to modify to create the DTR for various locations in the STDF log file.

Location of DTR	Test Program Sequence File Callback Sequence	Custom Result Processor Entry Point
At beginning of log (after initial records)	ProcessSetup	Model Plugin - Begin
After each batch (after last PRR of current batch and before first PIR of next batch)	PostBatch	Model Plugin - Batch Done
With each part using Sequential process model	PostUUT	Model Plugin - UUT Done

At end of log (before TSRs, HBRs, SBRs, and PCR)	ProcessCleanup	Model Plugin - Pre Batch Add the following precondition to the step: `!Parameters.ContinueTesting`

Adding DTRs for Wafer Testing When the Generate One Filer per Wafer Option is Enabled

If the Generate One File per Wafer option is enabled, the STDF Log result processor writes summary records (TSRs, HBRs, SBRs, and PCR) and closes the STDF file when it encounters the end of the wafer in the Model Plugin - Batch Done entry point. To ensure that DTRs are included in the wafer STDF file, you must add the DTRs before the STDF Log result processor Model Plugin - Batch Done entry point is called.

The custom result processor that you create must appear before the STDF Log result processor in the order of result processors in the Result Processing dialog box. The following table lists the sequences to modify to create the DTR for various locations in the STDF log file. In some cases, you must use a custom result processor because there is no test program callback sequence that generates the desired results.

Desired Location of DTR	Test Program Sequence File Callback Sequence	Custom Result Processor Entry Point
At beginning of each wafer log (after initial records) using Batch process model		Model Plugin - Pre Batch Execute step only if `SemiconductorModuleManager.BatchRuntimeData.IsStartOfWafer` is True.
At beginning of each wafer log (after initial records) using Sequential process model		Model Plugin - Pre UUT Execute step only if `SemiconductorModuleManager.BatchRuntimeData.IsStartOfWafer` is True.
With each batch (after last PRR of previous batch and before first PIR of current batch)	PreBatch	Model Plugin - Pre Batch or Model Plugin - Batch Done

With each part using Sequential process model	PreUUT	Model Plugin - Pre UUT or Model Plugin - UUT Done
With last batch before end of each wafer log using Batch process model		Model Plugin - Batch Done Execute step only if `SemiconductorModuleManager.BatchRuntimeData.IsEndOfWafer` is True
With last part before end of each wafer log using Sequential process model		Model Plugin - UUT Done Execute step only if `SemiconductorModuleManager.BatchRuntimeData.IsEndOfWafer` is True.

Lot Summary Reports (TSM)

When testing completes, you can generate a test Lot Summary Report that provides information about testing results. You can enable the Lot Summary Report in the Results Processing dialog box. The Lot Summary Report includes the following sections:

- **Lot Description Header**—Contains lot and station settings information and execution data, such as start and end time, that describe or identify the test cell, test conditions, state of Offline Mode, and DUTs tested.
- **Lot Results**—Contains results of all DUTs from all sites, including the percent-of-total yield. The Lot Results section also includes subsections for all tests and Inline QA only (if available) tests. You can use this data as a metric to judge the overall quality of the test lot.
- **Site Results**—Contains a table of test lot statistics by site. You can use this data as a metric to compare the test quality among different sites.
- **Software Bin Results**—Contains a table of software bin numbers, associated hardware bin numbers, software bin descriptions, total counts, and percent-of-total yield. You can use this data to identify specific types of failures reported during testing.

- **Hardware Bin Results**—Contains a table of hardware bin numbers, hardware bin descriptions (if available), total counts, and percent-of-total yield. You can use this data to separate DUTs with different failure modes or passing grades and to reconcile tester software counts with physical DUT counts.
- **Test Results**—Contains a table of test evaluation results by site for all the tests that executed at least once in the lot, sorted by execution order. You can use this data to compare the results of lots, which can be helpful during debugging. If alarms are enabled, the table includes the number of alarms raised when executing each test step.
- **Alarms**—If alarms are enabled, this section contains a table with the number of alarms of a given type that were raised during test program execution, on each site, pin, and step.

Note You can configure the Lot Summary Report in the Result Processing dialog box to display in the Report pane in the sequence editor and in the reports dialog box in the default operator interface.

See Also

Customizing the Lot Summary Report Header

Lot Summary Options Dialog Box

Customizing the Lot Summary Report Header (TSM)

The Lot Summary Report result processing plug-in sets the information in the Lot Summary Report header by using the values of the properties in the NI_SemiconductorModule_LotSettings and NI_SemiconductorModule_StationSettings data types.

You can modify the following station settings and lot settings values to customize the values the Lot Summary Report result processing plug-in sets in the Lot Summary Report header.

- **Station Settings**

- NodeName
- HandlerType

- **Lot Settings**
  - JobName
  - JobRevision
  - PartType
  - LotId
  - TestFlowId
  - TestTemperature
  - OperatorName

Debug Test Results Logs (TSM)

When testing a lot, you can generate a Debug Test Results Log of human-readable text that contains the measurement values and test limits for tests on all test steps that execute on each site. The Debug Test Results Log result processor generates a separate file for each site in the test program. You can use this data to debug test and chip design issues and to diagnose test issues on the tester itself.

Notes

- Enabling the Debug Test Results Log might affect the performance of a test program. Perform the following benchmark to ensure there is no performance loss when enabling the Debug Test Results Log: In the operator interface, enable the Debug Test Results Log and run the test program. When the test program execution completes, subtract the socket time from the cycle time displayed in the statistics indicator to calculate the tester index time. If the tester index time is less than handler/prober index time, enabling the Debug

Test Results Log will not impact performance.

- You can configure the Debug Test Results Logs in the Result Processing dialog box to display in the Report pane in the sequence editor and in the reports dialog box in the default operator interface.

The Debug Test Results Log contains the following sections for each DUT:

- **Header**—Contains the Site Number, Batch Number, Part ID, and state of Offline Mode for each DUT tested. The Batch Number refers to the loop iteration when testing multiple DUTs. The Debug Test Results Log result processing plug-in populates the Part ID field from the TestStand `SerialNumber` property, which TSM sets to the values returned in the **SitePartIds** parameter of the `StartOfTest` handler/prober entry point.

Note By default, if the handler/prober driver does not set the values in the **SitePartIds** parameter, TSM automatically assigns sequential numeric values to the `SerialNumber` property, which results in unique Part ID field values. To disable the default behavior, set the `GenerateUniquePartIds` property of the NI_SemiconductorModule_StationSettings data type to `False` in the ConfigureStationSettings or GetStationSettings callback sequence. When the `GenerateUniquePartIds` property is `True`, TSM reassigns the same unique Part ID to the part when it is retested. Customize the behavior of `GenerateUniquePartIds` to assign a new unique Part ID to a part when it is retested.

- **Step Results**—Contains the Step Name and all its corresponding tests.

- **Test Results**—Contains the Test Number, Result, Test Name, Low Limit, Measurement Value, Correlation Offset (if used), High Limit, and Units.

Modifying Number of Results to Include

By default, the Debug Test Results Log includes all test results. You can change the default behavior.

Complete the following steps to log results only when a DUT fails.

1. Launch the Debug Test Results Log Options dialog box.
2. Enable the **Log Results Only for DUT Failures** option.

Complete the following steps to limit the number of results to display for the Debug Test Results Log.

1. Launch the Debug Test Results Log Options dialog box.
2. Enable the **Limit Number of Results Displayed in Report View** option.
3. Use the **Display Results for Last** option to specify the number of tests to display in the Report View for the Debug Test Results Log.

Changing Report Orientation

By default, the Debug Test Results Log uses portrait orientation. Landscape orientation uses wider columns for tests with long test numbers or test names. Complete the following steps to change the report orientation of the Debug Test Results Log.

1. Launch the Debug Test Results Log Options dialog box.
2. Use the drop-down menu of the **Report Orientation** option to select **Portrait** or **Landscape**.

Customizing Filename

When you customize the log filename, use unique filenames for each test site. The Debug Test Results Log result processing plug-in appends results from each DUT to the corresponding site log.

Logging Text Data

If you want to add data to the Debug Test Results Log that is not a measurement or test limit, you can include text data in the Debug Test Results Log by using the following techniques:

- **Adding Text Data for a Step Using Additional Results**—You can use the Additional Results panel of the Step Settings pane to add text data after all the measurements and test limits for the tests associated with the step. To configure the Additional Results panel to generate text data in the Debug Test Results Log, set the **Name** option of the result to `NI.TestResultsLog` and set the **Value to Log** option to the value you want to add to the log file. You can optionally add a text label for the data by adding it to the end of the **Name** field in the form `NI.TestResultsLog.CustomTextLabel`, where CustomTextLabel is the text you want to display as a label in the log file.
- **Adding Text Data between Test Steps**—If the text data you want to log is not associated with a specific step, you can use the Additional Results step type to add text data to the Debug Test Results Log. Use the Additional Results panel of the Step Settings pane as described above to add data to the log file.

See Also

Debug Test Results Log Options Dialog Box

CSV Test Results Logs (TSM)

When a DUT completes testing, you can generate a CSV Test Results Log that contains data in a comma-separated values text file, which provides better performance than the Test Results Log result processor in a production environment. The CSV Test Results Log result processor generates a single file for all sites in the test program. You can open the `.csv` file directly in a spreadsheet application for analysis or to correlate test results.

Note You cannot view the CSV Test Results Log in the sequence editor or default operator interface.

The CSV Test Results Log contains the following sections for each DUT:

- Wafer ID—Enables the **Log Wafer Data** option in the CSV Test Results Log Options dialog box.
- Batch Number
- Site Number
- Part ID
- Die X Coordinate—Enables the **Log Wafer Data** option in the CSV Test Results Log Options dialog box.
- Die Y Coordinate—Enables the **Log Wafer Data** option in the CSV Test Results Log Options dialog box.
- Sequence Name
- Step Name
- Test Number
- Test Name
- Result
- Low Limit
- Value—The CSV Test Results Log does not scale data values.
- High Limit
- Correlation Offset
- Units
- Code Module Time—Enables the **Log Code Module Execution Time** option in the CSV Test Results Log Options dialog box.

Note The CSV Test Results Log does not include the state of Offline Mode.

Generating One File per Wafer

Enable the **Generate One File per Wafer** option on the CSV Test Results Log Options dialog box to create a log file for each wafer you test. Each new log file resets the batch number to `1`.

Modifying Number of Results to Include

By default, the CSV Test Results Log includes all test results. You can change the default behavior.

Complete the following steps to limit the number of results to include in the CSV Test Results Log.

1. Launch the CSV Test Results Log Options dialog box.
2. Enable the **Limit Number of Test Data Records** option.
3. Use the **Log test data for only one out of every** option to specify the number of tests to include in the CSV Test Results Log.

See Also

CSV Test Results Log Options Dialog Box

Configuring Handler or Prober Support for a Test Program (TSM)

Semiconductor testers often use handlers and probers to complete the following tasks:

- Place untested DUTs in test sites
- (Handlers) Move tested DUTs from test sites to an appropriate hardware bin, depending on the test results
- Potentially notify the tester to stop the test when no DUTs remain

The tester software must be able to communicate with the handler or prober to execute tests correctly. Handlers and probers use a variety of different communication protocols and command sets. Test program developers might need to ensure that the tester can use different types of handlers or probers.

TSM Implementation

Use the TSM handler/prober driver plug-in architecture to write and enable handler/prober driver sequence files. A handler/prober driver sequence file contains

handler/prober driver entry point sequences that TSM calls during execution to accomplish handler-related or prober-related tasks.

Use the NI Built-in Simulated Handler Driver to simulate handler functionality without requiring access to a real handler. To integrate a real handler or prober in the handler/prober driver plug-in architecture, you must create a new handler/prober driver sequence file. TSM uses only one active handler/prober driver sequence file at a time because the lot of DUTs (test lot) can use just one type of handler or prober at a time.

See Also

Specifying Settings for the Current Test Station

General Tab of Configure Station Settings Dialog Box

Creating a Handler/Prober Driver Sequence File (TSM)

Complete the following steps to use the `HandlerProberDriver.seq` file, located in the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory, as a starting point for a handler/prober driver sequence file you create.

1. Copy the `HandlerProberDriver.seq` file from the `<TestStand>\Components\Modules\NI_SemiconductorModule\Templates` directory to the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\HandlersAndProbers` directory.

   Note Create the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\HandlersAndProbers` directory if it does not already exist.

2. Rename the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\HandlersAndProbers\HandlerProberDriver.seq` file using the **<CompanyName>_<HandlerOrProberName>**`.seq` convention.
3. Open the handler/prober driver sequence file.

4. Modify each of the handler/prober driver entry point sequences to meet the requirements of the test system. A handler/prober driver sequence file must contain each handler/prober driver entry point except Configure. You can leave an entry point empty if you do not want the entry point to perform any functionality.
5. Save the handler/prober driver sequence file.

Handler/Prober Driver Entry Points (TSM)

TSM invokes a handler/prober driver by calling the following set of required entry point sequences, which specify a predefined set of parameters, at specific points during an execution:

- Configure—Provides a mechanism for end users to configure the handler/prober driver.
- Setup—Performs required handler or prober initialization tasks.
- StartOfTest—Determines when the handler or prober has placed untested DUTs in test sites by waiting for the handler start-of-test notification.
- EndOfTest—Sends the end-of-test notification to the handler or prober to move DUTs from test sites to hardware bins.
- Cleanup—Performs required handler or prober finalization tasks.

Configure Handler/Prober Driver Entry Point (TSM)

Use the Configure entry point to provide a mechanism for end users to configure the handler/prober driver and to persist the settings for multiple lots.

The General tab of the default Configure Station Settings dialog box contains a **Configure Handler/Prober** button that calls the Configure entry point of the handler/prober driver sequence file that the `Standard.HandlerDriverSequenceFilePath` property of the station settings specifies.

Parameters

The Configure entry point accepts the following parameters:

- **HandlerDriverData** [In/Out]—Container that stores handler-specific or prober-specific settings or run-time data. In a handler/prober driver sequence file, you can modify the default structure of this parameter, such as by changing the data type from the default Container to a custom container data type. However, the Configure entry point must create, and optionally assign, each field of this parameter using the TestStand API, such as the PropertyObject.NewSubProperty method or a `PropertyObject` Set value method, such as SetValNumber.

  Do not enable the **Check Type** option for this parameter. Right-click the parameter and remove the checkmark from **Check Type** in the context menu to disable the Check Type option.

- **Canceled** [Out]—Boolean value that notifies TSM whether the handler/prober settings have been modified and must be saved to disk. A value of `True` indicates that the handler/prober settings have not changed. A value of `False` indicates that the handler/prober settings have been modified and must be saved to disk.

- **SemiconductorModuleManager** [In/Out]—Object reference to an instance of a Semiconductor Module Manager. Use this object reference with the TSM Application API to get information about the currently configured lot.

Persisting Handler/Prober Configuration Settings

TSM automatically persists the configuration settings that the Configure entry point returns in the **HandlerDriverData** parameter to disk when the **Canceled** parameter is `False`. TSM stores configurations settings for all handler/prober drivers in the `<TestStand Config>\NI_SemiconductorModule\HandlerProberDrivers.cfg` file.

Setup Handler/Prober Driver Entry Point (TSM)

Use the Setup entry point to perform required handler or prober initialization tasks.

Parameters

The Setup entry point accepts the following parameters:

- **HandlerDriverData** [In/Out]—Container that stores handler-specific settings, prober-specific settings, or run-time data. The input value of this parameter is the same as the output value of the **HandlerDriverData** parameter of the Configure entry point. You can add run-time data to this container by using the TestStand API, such as the PropertyObject.NewSubProperty method or a `PropertyObject` Set value method, such as SetValNumber. TSM passes the output value of this parameter into the **HandlerDriverData** parameter of the StartOfTest, EndOfTest, and Cleanup entry points. TSM does not persist run-time data that you add to **HandlerDriverData** in the Setup entry point.

  Do not enable the **Check Type** option for this parameter. Right-click the parameter and remove the checkmark from **Check Type** in the context menu to disable the Check Type option.

- **StationSettings** [In/Out]—Instance of the NI_SemiconductorModule_StationSettings data type. The Setup entry point can access and modify the value of this parameter.

- **LotSettings** [In/Out]—Instance of the NI_SemiconductorModule_LotSettings data type. The Setup entry point can access and modify the value of this parameter. Wafer probers typically set the values of the properties in the `Standard.Wafer` container to set the fields in the Wafer Configuration Record (WCR) of the STDF log file.

- **SemiconductorModuleManager** [In/Out]—Object reference to an instance of a Semiconductor Module Manager. Use this object reference with the TSM Application API to get information about the currently configured lot.

StartOfTest Handler/Prober Driver Entry Point (TSM)

Use the StartOfTest entry point to determine when the handler or prober has placed untested DUTs in test sites by waiting for the handler start-of-test notification.

The StartOfTest entry point must notify TSM to continue or stop testing. When you execute using the Batch process model, the StartOfTest entry point must also notify TSM of the active or disabled state of each test site. When you execute using the Sequential or Parallel process model, TSM calls the StartOfTest entry point for each site independently.

Parameters

The StartOfTest entry point accepts the following parameters:

- **HandlerDriverData** [In/Out]—Container that stores handler-specific settings, prober-specific, or run-time data. In a handler/prober driver sequence file, you can modify the default structure of this parameter, such as by changing the data type from the default Container to a custom container data type. However, the Configure entry point must create, and optionally assign, each field of this parameter using the TestStand API, such as the PropertyObject.NewSubProperty method or a `PropertyObject` Set value method, such as SetValNumber.

  Do not enable the **Check Type** option for this parameter. Right-click the parameter and remove the checkmark from **Check Type** in the context menu to disable the Check Type option.

- **RequestedSiteState** [In]—Array of Boolean values that the StartOfTest entry point uses to determine the sites in which the handler must place DUTs or the prober should test. The index to the array corresponds to the site number of the test site.

  A value of `True` indicates that the handler must place a DUT in the site and that the handler driver must wait to receive the start-of-test notification for

that site. A value of `False` indicates that the handler must not place a DUT in the site and that the handler driver must not wait to receive a start-of-test notification for that site. For probers, a value of `True` indicates that the prober should test the die that corresponds to the test site.

When you execute using the Batch process model, TSM sets values in the array that correspond to sites being tested to `True`. When you execute using the Sequential or Parallel process model, TSM sets to `True` only the value in the array that corresponds to the current site.

- **ContinueTesting** [Out]—Boolean value the StartOfTest entry point uses to notify TSM to continue or stop testing. A value of `True` indicates that testing can continue because DUTs are in place. A value of `False` indicates that testing must stop. Set the value to `False` if the handler or prober indicates that it has no DUTs to test.
- **ActualSiteState** [Out]—Array of Boolean values the StartOfTest entry point uses to notify TSM of the active or disabled state of each test site on the tester. The index to the array corresponds to the site number of the test site. TSM uses this parameter only when you use the Batch process model to execute the test.

A value of `True` indicates that the tester must test the site. A value of `False` indicates that the tester must skip the site. For each site that the handler or prober indicates is active, set the corresponding array element to `True`. For each site that the handler or prober indicates is disabled, set the corresponding array element to `False`.

- **SitePartIds** [Out]—Array of strings that specifies the PART_ID field in the Part Results Records (PRR) of each part to test. The index to the array corresponds to the site number of the test site.
- **SitePartTexts** [Out]—Array of strings that specifies the PART_TXT field in the PRR of each part to test. The index to the array corresponds to the site number of the test site.
- **WaferRuntimeData**—Instance of the NI_SemiconductorModule_WaferRuntimeData data type, which is a container with the following properties:

- **StartOfWafer** [Out]—Boolean value that indicates whether the tester is starting to test a new wafer. Set this property to `True` when the prober indicates that it is starting a new wafer. TSM uses this property to determine when to generate Wafer Information Records (WIR) of the STDF file. When executing with the Batch process model, TSM also uses this property to determine the end of the previous wafer if the prober driver does not set the **EndOfWafer** parameter in the EndOfTest handler/prober driver entry point.
- **SiteDieCoordinates** [Out]—Array of NI_SemiconductorModule_WaferDieCoordinate data type that specifies the coordinates of each die to test. Set the elements in this array using the die coordinates the prober provides. Each element in the array contains the coordinates of one die. The index to the array corresponds to the site number of the test site. TSM uses the die coordinates to set the X_COORD and Y_COORD fields in the Part Results Records (PRR) of the STDF log file. Values for die coordinates must be in the range -32767 to 32767. By default, the die coordinates have the value -32768, which indicates that the coordinate is missing or unknown.
- **Identity**—Container of properties that identify the wafer, including the following components:

Note Set the `Identity` properties only when starting to test a new wafer. TSM uses the values in the `Identity` property only when the `StartOfWafer` property value is `True`.

  - **WaferId** [In/Out]—String that specifies the wafer ID to set the WAFER_ID field in the WIR of the STDF log file. The SemiconductorModule automatically generates unique wafer IDs, but the handler driver can override those IDs by setting this property.
  - **FabWaferId** [Out]—String that specifies the value to use for the FABWF_ID field in the WRR of the STDF log file.
  - **FrameId** [Out]—String that specifies the value to use for the FRAME_ID field in the WRR of the STDF log file.

- **MaskId** [Out]—String that specifies the value to use for the MASK_ID field in the WRR of the STDF log file.
- **UserDescription** [Out]—String that specifies the value to use for the USR_DESC field in the WRR of the STDF log file.
- **ExecDescription** [Out]—String that specifies the value to use for the EXC_DESC field in the WRR of the STDF log file.

Note TSM does not use the values returned in the **WaferRuntimeData** parameter when running with the Parallel process model.

- **SemiconductorModuleManager** [In/Out]—Object reference to an instance of a Semiconductor Module Manager. Use this object reference with the TSM Application API to get information about test execution, obtain test statistics, monitor the state of the test system, and so on.

See Also

Start of Test Dialog Box

EndOfTest Handler/Prober Driver Entry Point (TSM)

Use the EndOfTest entry point to send the end-of-test notification to move DUTs from test sites to hardware bins.

TSM sends the hardware bin results of the current DUTs to the EndOfTest entry point.

Parameters

The EndOfTest entry point accepts the following parameters:

- **HandlerDriverData** [In/Out]—Container that stores handler-specific settings, prober-specific settings, or run-time data. In a handler/prober driver

sequence file, you can modify the default structure of this parameter, such as by changing the data type from the default Container to a custom container data type. However, the Configure entry point must create, and optionally assign, each field of this parameter using the TestStand API, such as the PropertyObject.NewSubProperty method or a `PropertyObject` Set value method, such as SetValNumber.

Do not enable the **Check Type** option for this parameter. Right-click the parameter and remove the checkmark from **Check Type** in the context menu to disable the Check Type option.

- **HardwareBinData** [In]—Array of hardware bin numbers that specify the hardware bins assigned to each part on each site. The index to the array corresponds to the site number of the test site. A hardware bin value of `-1` indicates that TSM did not assign a hardware bin for the site for one of the following reasons:
  - When you execute tests using the Batch process model, you disabled the site.
  - When you execute tests using the Parallel process model and the site is not the currently executing site.
- **SoftwareBinData** [In]—Array of software bin numbers that specify the software bins assigned to each part on each site. The index to the array corresponds to the site number of the test site. A software bin value of `-1` indicates that TSM did not assign a software bin for the site for one of the following reasons:
  - When you execute tests using the Batch process model, you disabled the site.
  - When you execute tests using the Parallel process model and the site is not the currently executing site.
- **EndOfWafer** [Out]—Boolean value that indicates whether the tester just finished testing the last batch of parts on a wafer. Set this property to `True` when the prober indicates that there are no more die to test on the wafer. TSM uses this property to determine when to generate Wafer Results Records (WRR) of the STDF log file and to determine when to finish writing STDF logs

and Lot Summary Reports when you enable the Generate One File per Wafer option for those report generators.

This output parameter is optional when you execute with the Batch process model. If the prober driver does not set this parameter when executing with the Batch process model, TSM infers the end of the wafer when the prober sets the **WaferRuntimeData.StartOfWafer** parameter to true in the StartOfTest handler/prober driver entry point. When using the Batch process model, omitting the wait for an end-of-wafer status message to set the **EndOfWafer** parameter in the EndOfTest entry point might improve performance of the test system.

- **BinTypes** [In]—Array of numbers that specifies the types of the bins assigned to each part on each site. Each element in the array has one of the following values to indicate the type of bin of the corresponding element in the **SoftwareBinData** and **HardwareBinData** parameter arrays:
  - **0**—The bin is a Pass bin.
  - **1**—The bin is a Fail bin.
  - **2**—The bin is an Other bin.
  - **-1**—TSM did not assign a bin for the part tested on that site.
- **SemiconductorModuleManager** [In/Out]—Object reference to an instance of a Semiconductor Module Manager. Use this object reference with the TSM Application API to get information about test execution, obtain test statistics, monitor the state of the test system, and so on.

Note If one or more test sockets in a batch execution prematurely stop running, such as when you abort a test socket, TSM stops all tests without calling the EndOfTest entry point for the current DUTs.

See Also

End of Test Dialog Box

Cleanup Handler/Prober Driver Entry Point (TSM)

Use the Cleanup entry point to perform required handler/prober finalization tasks.

Parameters

The Cleanup entry point accepts the following parameters:

- **HandlerDriverData** [In/Out]—Container that stores handler-specific settings, prober-specific settings, or run-time data. In a handler driver sequence file, you can modify the default structure of this parameter, such as by changing the data type from the default Container to a custom container data type. However, the Configure entry point must create, and optionally assign, each field of this parameter using the TestStand API, such as the PropertyObject.NewSubProperty method or a `PropertyObject` Set value method, such as SetValNumber.

  Do not enable the **Check Type** option for this parameter. Right-click the parameter and remove the checkmark from **Check Type** in the context menu to disable the Check Type option.

- **SemiconductorModuleManager** [In/Out]—Object reference to an instance of a Semiconductor Module Manager. Use this object reference with the TSM Application API to get information about the currently configured lot.

Ensuring That a Handler or Prober Driver Captures All Start-of-Test Notifications (TSM)

A handler or prober driver coordinates a connected computer with the tester by using the driver entry point sequences to wait for a start-of-test (SOT) signal before beginning a test. Missing an SOT signal might cause the tester to wait indefinitely.

To ensure that a driver captures each SOT signal, the instrument communicating with the handler or prober must be armed for capturing the SOT signal before sending the end-of-test (EOT) signal. The handler or prober driver can capture the SOT signal if it occurs when the tester performs tasks during the tester index time or continue waiting until the SOT signal arrives.

Implement the following functionality in the driver entry points to ensure that the handler or prober driver captures all SOT signals:

Entry Point	Functionality to Implement
Setup	1. Open instrument sessions with which the handler or prober communicates. 2. Configure the instrument with any armed triggers or monitoring ports for capturing the SOT signal. 3. Query for any machine or lot information. 4. Send any initialization routine to the hand ler or prober, such as load first wafer.
StartOfTest	1. Determine whether the SOT signal was re ceived. 2. Continue monitoring for an end-of-lot (EO L) signal while waiting for the SOT signal. Check until the trigger arrives or the tester sends an EOL signal.
EndOfTest	1. Reconfigure the instrument to capture the next SOT signal. 2. Send the EOT signal, including any test re sults.
Cleanup	1. Send a signal to stop the handler or probe r. 2. Close instrument sessions.

NI Built-in Simulated Handler Driver (TSM)

When developing, testing, or debugging a semiconductor test program, the test developer might not have access to a real handler or prober. In such cases, the test developer can use the NI Built-in Simulated Handler Driver to verify that the test program behaves correctly without a handler or prober.

Note You can configure the NI Built-in Simulated Handler Driver to launch the Start of Test and the End of Test dialog boxes.

Enabling and Configuring the NI Built-in Simulated Handler Driver

Complete the following steps to enable and configure the NI Built-in Simulated Handler Driver.

Note The following steps describe the default TSM implementation of the Configure Station Settings dialog box.

1. (TestStand Sequence Editor) Select **Semiconductor Module»Configure Station** to launch the default Configure Station Settings dialog box.

   (Operator Interface) Click the **Configure Station** button.
2. In the **Enable Handler/Prober Driver (Real or Simulated)** section on the General tab of the Configure Station Settings dialog box, select **Built-in Simulated Handler Driver** in the **Handler/Prober Driver** option to enable the NI Built-in Simulated Handler Driver for the test program. TSM uses only one active handler driver sequence file at a time because the lot of DUTs (test lot) can use just one type of handler at a time.
3. Click **Configure Handler/Prober** to launch the Configure Built-in Simulated Handler dialog box.

Handler/Prober Modes (TSM)

TSM stores the handler/prober modes as one of the following numeric values for the `Standard.HandlerMode` property of the station settings:

Numeric Value	Description
0	Specifies no handler or prober interaction.
1	Specifies to use the NI Built-in Simulated Handler Driver to interact with dialog boxes to simulate the behavior of a handler and to view the test results.

2	Specifies to use a real or simulated handler/prober driver other than the NI Built-in Simulated Handler Driver.

Performing Tasks when Lot Testing Completes (TSM)

TSM calls the `LotTestingComplete` callback sequence to perform tasks when a lot completes testing, such as sending generated reports to a central server or displaying a message on the tester to indicate that the tester is idle.

The default implementation of the `LotTestingComplete` callback sequence is empty. You can override this callback to customize the tasks to perform when a lot completes testing. TSM calls the `LotTestingComplete` callback after all other process model plug-ins complete execution.

The `LotTestingComplete` callback sequence accepts the following parameters:

- **LotSettings** [In]—An instance of the NI_SemiconductorModule_LotSettings data type that contains the settings used during the lot that completed testing.
- **StationSettings** [In]—An instance of the NI_SemiconductorModule_StationSettings data type that contains the station settings used during the lot that completed testing.
- **ModelPluginConfiguration** [In]—An instance of the NI_ModelPluginConfiguration data type that contains the configuration and run-time variables for the set of all active process model plug-in instances. Use this parameter to extract information, such as report paths and directories.
- **ModelData** [In]—Contains information about the process model used for the lot that completed testing.

Customizing the LotTestingComplete Callback (TSM)

TSM calls the LotTestingComplete callback sequence, located in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq` or `<TestStand>\Components\Callbac`

ks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file, when a lot completes testing.

Complete the following steps to override the default LotTestingComplete callback sequence and customize the tasks to complete when a lot completes testing.

1. Determine whether a sequence file named SemiconductorModuleCallbacks.seq exists in the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule directory. If the sequence file does not exist, create it and ensure that it does not contain any sequences.
2. Copy the LotTestingComplete callback sequences from the <TestStand>\Components\Modules\NI_SemiconductorModule\Templates\SemiconductorModuleCallbacks.seq file to the <TestStand Public>\Components\Callbacks\NI_SemiconductorModule\SemiconductorModuleCallbacks.seq file and make changes to the copy.

Retesting a DUT (TSM)

You can initiate a retest from the TestStand Sequence Editor, an operator interface, or a handler/prober. The STDF log file includes retest information.

Manually Initiating a Retest from the TestStand Sequence Editor or Operator Interfaces

Click the **Retest** button on the TSM toolbar or in the Lot Statistics Viewer while execution is paused to manually initiate a retest. In a custom operator interface, you can create a button bound to the **PerformSinglePartRetest** command to perform the same functionality as the **Retest** button.

When you manually initiate a retest, TSM retests the DUT without calling the EndOfTest or StartOfTest callbacks in the handler/prober driver and discards the bin results of the previous test for the retested DUT.

Initiating a Retest from a Handler/Prober

A handler/prober can initiate a retest by sending a Part ID or part die coordinates that match a previously tested part. TSM detects this condition, tracks the test as a

retest of the previous matching DUT, discards the bin results of the previous test for that DUT, and does not increment the part counts.

STDF File Contents after Retesting a DUT

When you retest a DUT, the STDF file includes the PIR, PRR, PTR, and FTR records for every test run, including retests. The WRR, PCR, HBR, and SBR summary records include only the results of the last retest and do not include retested results. The TSR summary record includes the counts of all tests run, including retests.

Deploying TSM Test Programs

Complete the following steps to use the TestStand Deployment Utility to deploy a test program.

1. Complete the following steps on the Mode tab of the deployment utility:
    a. Select the **Deployable Image Only** option.
    b. Select the **Create new Full Deployment** option.

    NI recommends that you create a new Full Deployment, save the deployment to increment the **Deployment Version**, and store the deployment in a source code control system each time you create or update a deployment.
2. Complete the following steps on the System Source tab:
    a. Place a checkmark in the **From Directory** checkbox and specify the test program directory.
    b. Place a checkmark in the **Include Subdirectories** checkbox.
    c. In the **Location of Deployable Image** field, specify the location in which to save the deployable image.
3. Click the Distributed Files tab and select **Yes** to analyze the source files.
4. Click the Build Status tab to display the analysis results. Resolve any issues before completing the next step.
5. Complete the following steps on the Distributed Files tab:

   a. Select the files to include in the distribution. Use the **Distributed Files** pull-down menu to filter the display.
   b. Click the **LabVIEW Options** button to launch the LabVIEW VI Options dialog box, which you use to specify LabVIEW options.

6. In the LabVIEW VI Options dialog box, enable the following options in the Packed Project Library Options section of the dialog box and use the default values for the other options in this dialog box.
   - Output VIs to a Packed Project Library ()
   - Copy files from vi.lib before Build
   - Copy files from user.lib before Build
   - Copy files from instr.lib before Build

   Enabling these options increases the size of the distribution but allows the test program to more easily run in the LabVIEW Run-Time Engine (RTE).
7. Click **OK** in the LabVIEW VI Options dialog box.
8. Complete the following steps on the Distributed Files tab:
   a. Click the **Save As** button if this is the first time you are configuring a deployment for the test program to save the distribution configuration for the deployment. Click the **Save** button when you modify an existing deployment to increment the version number of the deployment.
   b. Click the **Build** button to build the distribution.
   c. Click the **Save** button to auto-increment the **Deployment Version** field on the Mode tab.
9. Complete the remaining tasks in the Deployment Process Overview to transfer the deployment to a tester and validate the deployment.

Consider the NI recommendations for installers, LabVIEW code modules, .NET code modules, and protections for test programs and test limits as you design and create deployments for a semiconductor test system.

See Also

Network-Based Deployment Mechanisms

Installer Settings for Deploying TSM Test Programs

Use the following recommended settings to create a simple installer with the deployment utility:

- Use the **Deployment Version** option on the Mode tab of the deployment utility to specify a version for the distribution. Increment the version with each release of the test program distribution.
- Do not enable the **Install TestStand Runtime** option or include any software in the Drivers and Components dialog box. Do not use the test program deployment to control the software installed on a semiconductor test station.
- Enable the **Do not Ask User for Installation Directory** option on the Installer Options tab of the deployment utility and set the **Default Installation Directory** option to install the test program in the same location on all test stations.

After you create a deployment, save the deployment specification file (`.tsd`) so you can use the same configuration to build newer versions of the installer or to create patches. Additionally, on the Build Status tab of the deployment utility, save the deployment build status log files to use to troubleshoot issues.

Deploying LabVIEW Code Modules with TSM Test Programs

Build LabVIEW code modules into packed project libraries for deployment by enabling the **Output VIs to a Packed Project Library** option in the LabVIEW VI Options dialog box of the deployment utility. Benefits of using packed project libraries include faster load and execution times, less disk space, and automatic version control for installers.

If you need to debug a test program on a production system, NI recommends that you build separate release and debug deployment installers. In the debug deployment, enable the **Enable Debugging** option in the LabVIEW VI Options

dialog box to create a debuggable version of the packed project library that includes LabVIEW block diagrams for the VIs the packed project library contains. When debugging is enabled, you can debug VIs in the library on a target system with the LabVIEW Development System installed. NI recommends debug versions of the packed project libraries only for debugging, not for production testing, because the debug versions have a negative performance impact and use more memory.

You cannot modify VIs in a packed project library, even when you enable the Enable Debugging option. Enable the **Include Source for Rebuilding Packed Project Libraries** option in the Packed Project Library Options dialog box to include the source VIs for the packed project library in the deployment if you need to modify the VIs and then rebuild the packed project library on a target system.

See Also

Deploying VIs in LabVIEW Packed Project Libraries

Editing VIs in LabVIEW Packed Project Libraries

Managing Versioned and Non-Versioned Files

Patching LabVIEW VIs

Processing LabVIEW Code Modules for Deployment

Troubleshooting LabVIEW Code Module Issues

Deploying .NET Code Modules with TSM Test Programs

NI recommends that you do not include the TSM Code Module API assembly (`NationalInstruments.TestStand.SemiconductorModule.CodeModuleAPI.dll`) or any referenced NI hardware assemblies in a test program deployment. Instead, use TSM or corresponding driver installer to install these components as part of the system configuration to help ensure a stable test system configuration.

If you need to debug a test program on a production system, NI recommends that you create separate release and debug deployments. In the debug deployment, include the code module assembly built with a debug configuration and the program database file (`<assembly_name>.pdb` file) that contains the debug symbols for the assembly. NI recommends debug versions of the assemblies only for

debugging, not for production testing, because the debug versions negatively affect performance and use more memory.

Protecting Test Programs and Test Limits from Editing and Viewing (TSM)

You can password protect sequence files to deter editing and viewing sequence files in the sequence editor and in operator interfaces. To protect test limits files, use the **Embed Test Limits File** option on the Test limits Files panel of the Test Program Editor to embed the external test limits files in the test program sequence file before you password-protect the test program sequence file.

Note NI does not recommend using passwords as the only way of protecting intellectual property.

Operator Interfaces (TSM)

Use the TSM default LabVIEW operator interface or STS Operator Tool interface to run tests and monitor test status. You can customize each TSM default operator interface to change the permissions, menu items, and controls an operator can access.

Note The TSM default operator interfaces do not support executing sequences that create additional executions.

▾ Default LabVIEW Operator Interface

The default LabVIEW operator interface contains the following features:

- **Configure Station**—Launch the Configure Station Settings dialog box.
- **Configure Lot**—Launch the Configure Lot Settings dialog box.
- **Open STS Maintenance Software**—Launch STS Maintenance Software.
- **Login/Logout**—Log in or out of the operator interface.
- **Exit**—Exit the operator interface.

- **Command Buttons**—Run, pause, or stop a single test or a test lot. You might notice a delay after you click the **Pause** button or the **End Lot** button because tests in progress must complete before pausing or ending the lot.
- **Statistics Indicator**—Displays statistical information for a lot execution, such as cycle time and socket time or site part counts.
- **Tester Status Message**—Displays the status of the tester.
- **Site Execution Data**—Displays statistical and status information for each site.
- **Active STS Software**—Displays the name of the active STS Software installed on the system.
  - If any component in the active STS Software is missing from the system, an error icon appears with a tooltip directing you to open STS Version Selector for more information.
  - If STS Version Selector is not installed, the installed TSM version is displayed.
- **Settings Table**—Displays station and lot settings for the test program. You can customize this table.
- **View Mid-Lot Summary**—Generate and display a Mid-Lot Summary test report.
- **View Reports**—Generate and display reports for the current lot.
- **Bin Table**—Displays information about the binning of DUTs for the current lot, such as the names of the soft bins, the associated hard bins, and the DUT counts for each site.

Default STS Operator Tool Interface

The default STS Operator Tool interface contains the following features:

- A drop-down menu with the following options:
  - **Configure Lot**—Launch the Configure Lot Settings dialog box.
  - **Configure Station**—Launch the Configure Station Settings dialog box.

- **View Mid-Lot Summary**—Generate and display a Mid-Lot Summary test report.
- **View Reports**—Generate and display reports or the current lot.
- **Open STS Maintenance Software**—Launch STS Maintenance Software.

- **Command Buttons**—Run, pause, or stop a single test or test lot. You might notice a delay after you click the **Pause** button or the **End Lot** button because tests in progress must complete before pausing or ending the lot.
- **System Status**—Displays the status of the system.
- **User Icon**—Displays the user. Log in or out of the operator interface. Use the User Icon to temporarily access administrative privileges and view additional Failure Analysis details by clicking the  button.
- **Yield**—Displays the socket, index, and cycle times.
- **Test Settings**—Displays station and lot settings for the test program.
- **Failure Analysis**—Displays detailed statistics for the lot or last 10 tests. Use the drop-down menu to specify which set of statistics to show. Use the  or  buttons to toggle between the list view and grid view data visualization layouts, respectively. In grid view, hover over each cell in the table to see a tooltip with additional information.

Running a Test from a Default TSM Operator Interface

Complete the following steps to test a lot using the TSM default LabVIEW operator interface or the STS Operator Tool interface.

▾ Default LabVIEW Operator Interface

1. Launch the operator interface.
2. When the Login dialog box launches, enter or select a username and enter the password.
3. Complete the following steps to configure the lot.
    a. Click **Configure Lot**.

   b. Enter the lot information in the Configure Lot Settings dialog box and click **OK**.

4. Click **Start Lot** to begin testing at lot. Use the **Pause** and **Resume** buttons to pause the lot between DUTs. To test a single batch of DUTs and automatically pause after testing of those DUTs completes, click **Single Test**.
5. Click **End Lot** to stop testing the lot.
6. Repeat steps 3–5 to test a new lot.
7. Click **View Mid-Lot Summary** to generate and display a Mid-Lot Summary test report. You can view, refresh, and print the report any time during or after testing.
8. Click **View Reports** to view reports for the current lot any time during or after testing.
9. Click the **Exit** button to close the test application.

▾ Default STS Operator Tool Interface

1. Launch the operator interface.
2. When the Login dialog box launches, enter or select a username and enter the password.
3. Complete the following steps to configure the lot.

   a. Select **Configure Lot** from the drop-down menu.
   b. Enter the lot information in the Configure Lot Settings dialog box and click **OK**.

4. Click **Start Lot** to begin testing at lot. Use the **Pause** and **Resume** buttons to pause the lot between DUTs. To test a single batch of DUTs and automatically pause after testing of those DUTs completes, click **Single Test**.
5. Click the **End Lot** button to stop testing the lot.
6. Repeat steps 3–5 to test a new lot.
7. Select **View Mid-Lot Summary** from the drop-down menu to generate and display a Mid-Lot Summary test report. You can view, refresh, and print the report any time during or after testing.

8. Select **View Reports** from the drop-down menu to view reports for the current lot any time during or after testing.
9. Click the **X** button to close the test application.

Customizing Operator Interfaces (TSM)

Complete the following steps to customize the default LabVIEW operator interface or STS Operator Tool interface.

1. Copy the contents of the <TestStand>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp> directory to the <TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp> directory.

   Note Create the <TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp> directory if it does not already exist.

2. Modify the files in the <TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW or CSharp> directory to meet your requirements.

The operator interfaces use the TSM Application API extensively. Review the LabVIEW operator interface architecture to better understand how the LabVIEW source code is structured.

Notes If you are upgrading from an earlier version of TSM and use a custom operator interface based on TSM 2013 or earlier operator interface source code, you must re-implement the custom operator interface to use the latest TSM operator interface source code, which includes improvements for performance, customization, and maintenance. TSM 2017 removed support for custom operator interfaces that are based on the default TSM 2013 LabVIEW operator interface.

Customizing the Settings Table (TSM)

The settings table on the right side of the default operator interface displays common lot and station settings. You can configure the list of settings to display in the table by editing the `OISettingsTable.cfg` file located in the `<TestStand Application Data>\Cfg\NI_SemiconductorModule` directory. The `<TestStand>\Components\Schemas\NI_SemiconductorModule\OISettingsTable.xsd` schema file describes the format of the configuration file.

The `SemiconductorModuleManager` class in the TSM Application API uses the `OISettingsTable.cfg` configuration file to determine the list of settings the `GetSettingsToDisplay` method returns. All default TSM operator interfaces use the `GetSettingsToDisplay` method to populate the settings table.

Note The default operator interface reads the `OISettingsTable.cfg` file only at startup. You must restart the operator interface each time you make a modification to the `OISettingsTable.cfg` file.

Operator Interface Settings Table File XML Structure (TSM)

The operator interface settings table XML schema, located at `<TestStand>\Components\Schemas\NI_SemiconductorModule\OISettingsTable.xsd`, defines the following structure for an operator interface settings table configuration file:

Legend	
R	**<Root element>**
E	**<Element>**
A	`Attribute`

R <OISettingsTable>

- (E) **<Settings>**—Specifies the list of settings to display in the table. The order of the items in this list determines the order in which the settings appear in the table.
  - (E) **<Setting>**—Specifies a setting to display in a row in the table.
    - (A) `label`—Label to display in the left column of the table. TSM localizes the label using the strings in the `[NI_SEMICONDUCTORMODULE_OI_SETTINGS_TABLE]` section of a language resource file.
    - (A) `valueExpr`—TestStand expression TSM evaluates to determine the value to display in the right column of the table. The expression can refer to LotSettings and StationSettings variables to access the current lot settings and station settings.
    - (A) `visibleExpr`—(Optional) TestStand expression TSM evaluates to determine whether to display the setting in the table. The expression can refer to LotSettings and StationSettings variables to access the current lot settings and station settings. If this attribute is missing, TSM uses a default value of `true`.
    - (A) `displayFileName`—(Optional) Boolean value that indicates whether the settings table displays a simple filename when the `valueExpr` expression evaluates to an absolute path. If this attribute is missing, TSM uses a default value of `false`.

Displaying Specific Site Numbers in Operator Interfaces (TSM)

You can disable specific sites or use the particular connections of a pin map that match the DIB for the test station.

When you disable sites in the default Configure Lot Settings dialog box or use the `AvailableSiteNumbers` property on the NI_SemiconductorModule_StationSettings data type to specify which site numbers from a pin map for a test program to use when running the test program, the default TSM operator interfaces display only the sites you specify.

TSM 2016 and earlier default operator interfaces and custom operator interfaces based on those versions display site numbers starting at 0 and increasing by 1, up to the number of sites. You must make the following changes to custom LabVIEW or C# operator interfaces based on the TSM 2016 and earlier operator interfaces to display

the configured site numbers when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting:

- Query the Semiconductor Module Manager to retrieve information about which site numbers the running test program includes.
- Update the site status labels to show the correct site numbers.
- Update the bin table to show the correct site numbers.
- Update code to avoid an error in the case that the Semiconductor Module Manager is using 0 sites, which can happen only when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting.

Displaying Specific Site Numbers in Custom LabVIEW Operator Interfaces (TSM)

You must complete the following steps to modify custom LabVIEW operator interfaces based on TSM 2016 and earlier operator interfaces to display the configured site numbers when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting. Refer to the LabVIEW source code for the operator interface for an example.

1. Open a copy of the `LVSemiOI.lvproj` project in the `<TestStand Public>\UserInterfaces\NI_SemiconductorModule\<LabVIEW>` directory.
2. Complete the following steps to update the site status labels to show the correct site numbers.
    a. Open the `Controls/Site Label.lvclass/Site Label.ctl` control.
    b. Change the label for the numeric in the private data from `Site Number` to `Site Index` to more accurately reflect what it represents.
    c. Complete the following steps to add a VI to update the site label text for a site number.

a. Right-click the `Site Label.lvclass` and select `New::VI From Dynamic Dispatch Template` from the context menu.
b. Add an `Operator Interface State.lvclass` control and a numeric control (for `Site Number`)
c. Use the Read Engine VI to get the engine reference from the `Operator Interface State` wire.
d. Use an ActiveX Invoke Node to call the `GetResourceString` method on the engine reference. Pass in `NI_SEMICONDUCTOR_OPERATOR_INTERFACE` for the category and `SITE_NUMBER` for the symbol.
e. Wire the **Site Number** input to a Number to Decimal String node.
f. Use a Search and Replace String node by using the result from the `GetResourceString` method as the input string and replacing `%1` with the result from the Number to Decimal String node.
g. Use the Read Control Refnum VI to get the control refnum from the **Site Label in** input.
h. Use a Property Node to set the `Value` property of the control refnum to the resulting string from the Search and Replace String node.
i. Set up proper error wiring for the VI and save it as `Update Site Label Text.vi`.

d. Complete the following steps to add a VI to update the site number for a site label.

a. Right-click the `Site Label.lvclass` and select `New::VI From Dynamic Dispatch Template` from the context menu.
b. Add an `Operator Interface State.lvclass` control.
c. Use the Read Semiconductor Module Manager VI to get the manager reference from the operator interface state.
d. Use the Get Site Numbers VI from the TSM Application API to get the current set of site numbers from the manager reference.

e. Use the Unbundle By Name node to get the `Site Index` from the **Site Label** input.
f. Use Index Array on the array of site numbers using the `Site Index` as the index.
g. Pass the result from the **Index Array** to the Update Site Label Text VI you created previously.
h. Set up proper error wiring for the VI and save it as `Update Site Number.vi`.

e. Complete the following steps to add a call to the Update Site Number VI in the Refresh Callback VI for the site label.

a. Open the `Controls/Site Label.lvclass/Refresh Callback.vi`.
b. After the call to the Display Site Control VI, insert a call to the Update Site Number VI you created previously.

3. Complete the following steps to update the bin table to show the correct site numbers.

a. Open the `Controls/Bin Table.lvclass/Refresh Callback.vi`.
b. Copy the **Site Lot Statistics** array control from the front panel.
c. Open the `Controls/Bin Table.lvclass/Get Column Headers.vi` and complete the following steps.

a. Paste the **Site Lot Statistics** array control on the front panel.
b. Wire the **Site Lot Statistics** array to the right-most For Loop and enable auto-indexing.
c. Remove the wire from the **Number of Sites** input that connects to the **Count** terminal of the For Loop.
d. Inside the For Loop, wire the Lot Statistics reference to an ActiveX Property Node to retrieve the `SiteNumber` property.
e. Delete the wire from the For Loop iteration node to the Number to Decimal String node.

f. Wire the result from the `SiteNumber` property to the Number to Decimal String node.

g. Delete the **Number of Sites** control.

h. Make the **Site Lot Statistics** array control a required input on the connector pane.

d. In the Refresh Callback VI, which you should already have open, delete the Array Size node and wire the **Site Lot Statistics** array directly to the Get Column Headers call.

4. Complete the following steps to update the code to avoid an error in the case that the Semiconductor Module Manager is using 0 sites, which can happen only when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting.

   a. Open the `Controls/Last N Parts Label.lvclass/Refresh Callback.vi`.

   b. Delete the **Index Array**, the numeric constant node, and any connected wire segments.

   c. Wire the `All Site Lot Statistics` reference to the property node for PartCountWindowSize.

Displaying Specific Site Numbers in Custom C# Operator Interfaces (TSM)

You must complete the following steps to modify custom C# operator interfaces based on the TSM 2016 and earlier operator interfaces to display the configured site numbers when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting. Refer to the C# source code for the operator interface for an example.

1. Open a copy of `MainForm.cs` in the `<TestStand Public>\UserInterfaces\NI_SemiconductorModule\<CSharp>` directory.

2. Complete the following steps to query the Semiconductor Module Manager to retrieve information about which site numbers the running test program includes.
    a. In the `// Bin table member variables` section of the file, add a new field to store the site numbers from the Semiconductor Module Manager.
    b. In the `// Stop the lot statistics refresh timer while reconfiguring the controls` section of the file, initialize the site numbers from the Semiconductor Module Manager.
    c. Change the existing `// Reconfigure the controls if the number of sites has changed` section of the file to detect if the site number information has changed in the Semiconductor Module Manager, update the operator interface information about the number of sites and the site numbers, and reconfigure the operator interface controls.
3. To update the site status labels to show the correct site numbers, when you set the label text for the site in the `// Site Label column` section of the file, use the site index to get the appropriate site number from the stored site numbers of the operator interface.
4. Complete the following steps in the `// Add a column for each site` section of the file to update the bin table to show the correct site numbers.
    a. (Optional) Rename the existing `site` variable for iterating over the sites to `siteIndex` to clarify that the variable is the index into the set of sites and not the site number itself.
    b. When you set the site column header text, use the site index to get the appropriate site number from the stored site numbers of the operator interface.
5. Complete the following steps to update the code to avoid an error in the case that the Semiconductor Module Manager is using 0 sites, which can happen only when you disable sites in the default Configure Lot Settings dialog box or use the AvailableSites station setting.

a. In the `// Check if PartCountWindowSize has changed` section of the file, use the `AllSiteLotStatistics` property to get the PartCountWindowSize instead of querying the siteLotStatistics array to get the PartCountWindowSize because the AllSiteLotStatistics object exists and is valid even when you use 0 sites.
b. In the `// Update part count window size if it has changed` section of the file, remove the now unneeded parameter to the `ConfigurePartCountWindowSizeIfChanged` method and remove the comment that refers to the parameter.
c. In the `// Check if the SiteStatusIndicator controls have been drawn by checking if the width has been set` section of the file, add a check to ensure that any site part count statistics controls exist before querying the property from the first one.

Handling Errors in Operator Interfaces (TSM)

You can configure TSM operator interfaces to write a message to an error log file when any run-time error is raised and execute the behavior you specify in the SemiconductorModuleManager when a code module run-time error is raised.

To specify how TSM treats run-time errors in TSM Operator interfaces, complete the following steps:

1. Select **Configure»Station Options** to launch the Station Options dialog box.
2. Select **Show Dialog Box** from the **On Run-Time Error** drop-down menu.
3. Click **OK**.
4. Optional. Use the following TSM Application API property and event to customize error handling for the operator interface:
   - **ErrorLogFilePath**—Specifies the location of the error log file. The default setting for this property is `<TestStand Public>\ErrorLogs\NI_SemiconductorModule\OperatorInterfaceErrors.log`
   - **ErrorOccurred** event—Generated when any error occurs.

Code module run-time errors are a class of run-time errors. A **code module** run-time error is raised when the `MainSequence` sequence returns a run-time error from a source other than TSM, such as a code module or an instrument driver.

To specify how TSM treats code module run-time errors in TSM Operator interfaces, set the following TSM Application API properties on the SemiconductorModuleManager object in the operator interface source code:

- **EndLotOnCodeModuleRuntimeError**—Determines whether testing ends immediately after the `MainSequence` sequence returns a code module run-time error. The default setting for this property is `False`.
- **DisplayDialogOnCodeModuleRuntimeError**—Determines whether the operator interface displays the run-time error dialog box when the `MainSequence` sequence returns a code module run-time error. The default setting for this property is `False`.

Note By default, when any run-time error occurs, TSM assigns the current part to the Default Error bin the bin definitions file specifies.

LabVIEW Operator Interface Architecture (TSM)

The `LVSemiOI.lvproj`, located in the `<TestStand>\UserInterfaces\NI_SemiconductorModule\LabVIEW` directory, includes the `TestStand Semiconductor Module Operator Interface.lvlib`, which is the primary library for the operator interface.

See Also

High-Level Classes

Top-Level VI Overview

Updating Controls

Top-Level VI Overview (TSM)

The Top-Level VI includes the front panel operators that use the majority of the source code. This VI contains three main sections of code:

- **Initialization**—Configures and initializes the TestStand manager controls, connects and initializes all the controls on the front panel with the TSM Application API, localizes all the controls that contain localizable text, and starts the Application Manager control. Add new controls and indicators to this section or remove existing controls and indicators.
- **Execution**—The event structure contains a set of events for command buttons value changes, refreshing the front panel controls, and application exit. The While Loop that contains the event structure stops when the application exits. If the application generates a refresh controls event or the timeout of the event structure elapses, the While Loop iterates and the Do Refresh Controls VI executes, which causes each control to update.
- **Shutdown**—Cleans up all resources created during execution, closes the front panel if it is an executable, and shuts down and closes the application.

High-Level Classes (TSM)

The TSM LabVIEW operator interface uses the following high-level class to hold data the operator interface uses and to make customization easier.

- **Operator Interface State**—Holds references and keeps track of state information for the execution of the operator interface, including the following examples:
  - An array of references to the controls on the front panel
  - A reference to events that command buttons generate and that the event structure in the main execution loop of the Top-Level VI handles
  - TestStand manager controls
  - TestStand Engine
  - Custom events that notify the main execution thread of changes
  - A reference to the Semiconductor Module Manager object, a component of the TSM Application API that manages most of test execution

See Also

Command Button Class

Control Class

Control Class (TSM)

The Control class is the base class for many of the controls displayed on the front panel of the operator interface. You can override the Refresh Callback VI in the Control class to update the appearance of controls when the operator interface changes state, such as updating the counts on a statistics control at a fixed interval.

The following classes inherit from the Control class to display additional information in the operator interface:

- **Bin Table**—Displays information about the binning of DUTs of the current lot, such as the names of the soft bins, the associated hard bins, and the DUT counts for each site.
- **Command Button**—Binds a LabVIEW button to a command object in the TSM Application API, such as Start Lot, End Lot, Login/Logout, or Configure Lot. The command object handles the functionality of the button when a user clicks the button and updates the enabled state and text of the button when the state of the command changes.
- **InlineQA Label**—Displays a label for the statistics and status of inline QA. The label and statistics controls are hidden when inline QA is disabled.
- **Last N Parts Label**—Displays a label for the column of site status controls and the number of DUTs included in the site status.
- **Site Label**—Displays a label for the statistics and status of a particular site.
- **Site Status**—Displays a color-coded status of the most recent **n** DUTs that have been tested, where the Site Status Part Count station setting specifies **n**. The red portion of the bar indicates the number of DUTs that failed. The green portion of the bar indicates the number of DUTs that passed. The ratio of the two colors in the bar indicates the recent yield of DUTs tested in a particular site.

- **Site Testing Icon**—Displays an icon that indicates whether the handler has placed a DUT in the site or the prober is testing on the site.
- **Statistics Control**—Displays statistical information for a lot execution, such as cycle time and socket time or site part counts. When binding this control, specify the site and statistics type to display.
- **Test Program Info**—Displays the value of relevant station settings and lot settings for the loaded test program. The `OISettingsTable.cfg` configuration file determines the list of settings to display and how to format the settings values.
- **Tester Status**—Displays the current status of the tester, such as if a lot is testing or if any errors have occurred.

Command Button Class (TSM)

The Command Button class binds the buttons displayed on the front panel of the default operator interface to command objects in the TSM Application API. This class inherits from the Control class.

The TSM Application API includes the following types of commands:

- **Configure Lot**—Launches a dialog box to configure a lot.
- **Configure Station**—Launches a dialog box to configure the station.
- **Open STS Maintenance Software**—Launches STS Maintenance Software 17.1 or later.
- **Perform Single Part Test**—Starts a lot and tests a single DUT for each site if no lot is active and pauses between DUTs when complete or, if paused between DUTs, tests a single DUT for each site before pausing again.
- **Start Lot**—Starts testing a new lot.
- **Start/Resume Lot**—Starts testing a new lot, resumes a suspended sequence execution at a breakpoint, or resumes a sequence execution that is paused between DUTs.
- **Pause Lot**—Pauses testing of the lot. Testing pauses between DUTs after completing the tests for the current DUTs on each site and before TSM sends the end-of-test (EOT) signal to the handler or prober.

- **Pause/Resume Lot**—Pauses execution of the lot after the current DUT for each site completes testing, resumes a suspended sequence execution at a breakpoint, or resumes a sequence execution that is paused between DUTs. If the sequence execution is suspended at a breakpoint, you cannot use this command to pause the execution of the lot.
- **Retest**—After a single test completes or when you pause a lot, retests a single DUT for each site for the active sequence file and then pauses execution.
- **End Lot**—Ends execution of the lot.
- **View Mid Lot Summary**—Generates a Mid-Lot Summary text report and displays the report in a floating panel.
- **View Reports**—Displays the reports for the current execution in a floating panel.
- **Login/Logout**—Logs an operator in or out.
- **Exit**—Closes the operator interface.

Updating Controls (TSM)

Because LabVIEW operates on a data flow model and TestStand generates events, updating the state of execution while handling events might become a complex task. However, the Semiconductor Module Manager object handles most of these events and state management for you. The TSM default operator interface uses the `Command Button.lvclass` class to bind LabVIEW button controls to commands in the TSM Application API. The command objects generate events the Command Button uses to update the state of the button.

For all other controls, the main execution loop of the Top-Level VI calls the Refresh Callback VI of its connected class each iteration and updates the state of the control using information the TSM Application API provides.

TSM Sequence Editor UI Configuration

The default TSM UI Configuration of the TestStand Sequence Editor includes the following changes to streamline semiconductor test program development.

- Simplified toolbar

- Altered **Execute** menu that replaces the Test UUTs and Single Pass items with the Start Lot and Single Test items
- Modified Insertion Palette pane that displays the Semiconductor Module folder and the Action Step at the top of the Step Types list
- More detailed Steps pane

When you launch TSM for the first time or enable TSM, it loads the TSM UI Configuration, named `NI_SemiconductorModule`, and saves the most recently active UI configuration as `NI_SemiconductorModule_SavedLayout`. When you disable TSM, TestStand loads the `NI_SemiconductorModule_SavedLayout` UI configuration. You can modify the TSM UI Configuration and restore it to the default state.

Modifying the TSM UI Configuration

Complete the following steps to modify the TSM UI Configuration.

1. Customize the toolbars and menus and arrange the sequence editor panes to create the layout you want.
2. Select **Configure»Sequence Editor Options** and click the UI Configuration tab.
3. Select **NI_SemiconductorModule** in the **Saved Configurations** list and click the **Save Current** button to overwrite the existing TSM UI Configuration with the UI configuration you just created and to ensure TSM loads the modified UI configuration when you launch TSM for the first time or enable TSM.

Note When you first launch the sequence editor after installing TSM or when you enable TSM, TSM loads only the UI configuration named `NI_SemiconductorModule`.

Restoring TSM UI Configuration to Default State

Complete the following steps to restore the default TSM UI Configuration

1. Select **Configure»Sequence Editor Options** and click the UI Configuration tab.
2. Select **NI_SemiconductorModule** in the **Saved Configurations** list and click the **Delete Selected** button to delete the UI configuration.
3. Select **Semiconductor Module»Disable Semiconductor Module** to disable TSM.
4. Select **Semiconductor Module»Enable Semiconductor Module** to re-enable TSM, which loads the default TSM UI Configuration **NI_SemiconductorModule**.

Semiconductor Module Toolbar Buttons

The TSM toolbar contains the following buttons.

Note Use the TSM toolbar buttons to control execution and view lot statistics while debugging a sequence.

Command Name	Icon	Description
Edit Test Program: <**filename**>		Launches the Test Program Editor in which you can edit the test program settings for the active sequence file.
Edit Pin Map File		Opens the pin map file associated with the active sequence file in the Pin Map Editor.
Edit Bin Definitions File		Opens the bin definitions file in the Bin Definitions Editor.
Configure Station		Launches the Configure Station Settings dialog box.
Configure Lot		Launches the Configure Lot Settings dialog box.
Active Configuration		The test program configuration to use when testing. The available items correspond to the configurations in the active sequence file. The value initially corresponds to the value of the `LotS`

		ettings.Standard.ActiveConfigurationName property in the lot settings. If the active sequence file does not contain a configuration that corresponds to the ActiveConfigurationName lot setting, the control displays one of the configurations in the sequence file. Changing the selected configuration with this control does not modify the ActiveConfigurationName lot setting. You can also use the Configure Lot Settings dialog box to change the test program configuration.
Import Test Limits into <**filename**>		Imports Semiconductor Multi Test step test data into the active sequence file from an external test data file.
Export Test Limits from <**filename**>		Exports Semiconductor Multi Test step test data from the active sequence file to an external test data file.
Single Test		Starts a lot and tests a single DUT for each site for the active sequence file if no lot is active and pauses between DUTs when complete or, if paused between DUTs, tests a single DUT for each site before pausing again. Testing pauses between DUTs after completing the tests for the current DUTs on each site and before TSM sends the end-of-test (EOT) signal to the handler or prober.
Start/Resume Lot		Starts testing a new lot, resumes a suspended sequence execution at a breakpoint, or resumes

		a sequence execution that is pa used between DUTs.
Pause		Pauses testing of the lot. Testin g pauses between DUTs after co mpleting the tests for the curre nt DUTs on each site and before TSM sends the end-of-test (EOT) signal to the handler or prober .
Retest		After a single test completes or when you pause a lot, retests a single DUT for each site for the active sequence file and then p auses execution. TSM does not communicate with the handler or prober when you use the **Ret est** button.
End Lot		Ends testing the current lot at a ny time by stopping testing and closing instrument sessions by calling the ProcessCleanup mo del callback sequence.
Step Into		Enters and suspends within a fu nction, VI, or sequence the step calls. If the step calls a code mo dule TestStand cannot suspend within, TestStand suspends the execution at the next step.  **Note** When yo u step into a VI from TestStan d and then sel ect **Return to Cal ler** without execut ing the VI, any values you cha nge in the cont rols or indicato

		rs of the suspe nded VI are not returned to Te stStand.
Step Over		Executes the step to which the execution pointer points when t he sequence execution is in a br eakpoint state. If the step is a S equence Call step to another se quence, Step Over executes the entire sequence. The execution then enters a breakpoint state o n the step following the Sequen ce Call step. If the engine encou nters a breakpoint within the S equence Call step, the executio n pauses at the breakpoint.
Step Out		Resumes execution through the end of the current sequence an d suspends/pauses on the next step in the calling sequence.
Show Lot Statistics Viewer		Opens the Lot Statistics Viewer window to control execution an d view lot statistics while debug ging a sequence.
Show Runtime Data Viewer		Opens the Runtime Data Viewer window so you can see test res ults and debug issues at runtim e.
Launch Digital Pattern Editor		Launches the Digital Pattern Edi tor and opens the digital patter n project file, if any, associated with the active sequence file.
Launch InstrumentStudio		Launches InstrumentStudio, w hich is a pin- and site-aware, sof tware-based front panel applic ation you can use to monitor, c ontrol, and record measuremen ts from supported devices.

		Note If you launch InstrumentStudio in any other way, such as from the Microsoft Windows Start menu, InstrumentStudio is not pin and site aware.
Enable Offline Mode		Indicates Offline Mode is disabled. Click to enable Offline Mode, which allows you to develop, run, and debug test programs only on a computer without access to NI instruments.
Disable Offline Mode		Indicates Offline Mode is enabled. Click to disable Offline Mode.

Semiconductor Module Menu

The **Semiconductor Module** menu in the TestStand Sequence Editor contains the following items:

- **Edit Test Program: <filename>**—Launches the Test Program Editor, in which you can edit the test program settings for the active sequence file.
- **Edit Pin Map File**—Opens the pin map file associated with the active sequence file in the Pin Map Editor.
- **Edit Bin Definitions File**—Opens the bin definitions file associated with the active sequence file in the Bin Definitions Editor.
- **Create Test Program from Digital Pattern Project**—Launches the Create Test Program from Digital Pattern Project dialog box from which you can create a basic test program that initializes sessions for NI-Digital and NI-DCPower instruments and bursts the patterns defined in the digital pattern project.
- **Configure Station**—Launches the Configure Station Settings dialog box.

- **Configure Lot**—Launches the Configure Lot Settings dialog box.
- **Import Test Limits into <filename>**—Imports Semiconductor Multi Test step test data into the active sequence file from an external test data file.
- **Export Test Limits from <filename>**—Exports Semiconductor Multi Test step test data from the active sequence file to an external test data file.
- **Export Correlation Offset Template file based on <filename>**—Generates a tab-delimited correlation offsets template file based on the numerical limit tests in the selected sequence file.
- **Show Lot Statistics Viewer**—Opens the Lot Statistics Viewer window, in which you can control execution and view lot statistics while debugging a sequence.
- **Show Runtime Data Viewer**—Opens the Runtime Data Viewer window, in which you can see test results and debug issues at runtime.
- **Launch InstrumentStudio**—Launches InstrumentStudio, which is a pin- and site-aware, software-based front panel application you can use to monitor, control, and record measurements from supported devices.

Note If you launch InstrumentStudio in any other way, such as from the Microsoft Windows Start menu, InstrumentStudio is not pin and site aware.

- **Launch Digital Pattern Editor**—Launches the Digital Pattern Editor and opens the digital pattern project file, if any, associated with the active sequence file.
- **Custom Instrument Panels**—Displays and launches available custom pin- and site-aware instrument panels you can use to debug instruments during test program execution at a breakpoint.
- **Measure Performance of <filename>**—Launches the Test Program Performance Measurement Configuration dialog box, in which you can specify settings to execute and measure test program performance.
- **Launch Test Program Performance Analyzer**—Launches the Test Program Performance Analyzer so you can view data TSM generates when you measure the performance of a test program.

- **Enable/Disable Offline Mode**—Enables and disables Offline Mode, which allows you to develop, run, and debug test programs only on a computer without access to NI instruments.
- **Disable/Enable Semiconductor Module**—Disables and enables TSM and removes TSM type palettes and process model plug-ins.
- **About TestStand Semiconductor Module**—Displays version information for TSM.

TSM Steps Pane

Sequence File Window

The TSM Steps pane is similar to the TestStand Steps pane of the Sequence File window and contains the following default columns:

- **Step**—Displays the name of the step and the step icon. Click to the left of the step icon to toggle the breakpoint for the step. If you add a comment for a step, the comment appears in light text above the step name.
- **Description**—Displays a description of the step that varies according to the type of step and the adapter with which it was created. Includes the VI name or ClassName:MethodName for the associated code module.
- **Num Tests**—For a Semiconductor Multi Test step, displays the number of tests defined for the step.
- **Pins**—For Semiconductor Action and Multi Test steps, displays the pins selected on the Options tab of the step.
- **Multisite Option**—For Semiconductor Action and Multi Test steps, displays the Multisite Option selected on the Options tab of the step.
- **Settings**—Displays the properties of the step that contain non-default values. Use the Step Settings pane to specify step settings.

Execution Window

The TSM Steps pane is similar to the TestStand Steps pane of the Sequence File window and contains the following default columns:

- **Step**—The name and icon of the step. Click in the space to the left of the step icon to toggle the breakpoint for the step.
- **Description**—A description of the step that varies according to the type of step and the module adapter that it uses.
- **Settings**—The properties of the step that contain non-default values.
- **Module Time**—The code module time for the step for the cycle of execution.
- **Status**—The value of the status property for the step. If the step has not yet executed, the status is an empty string. After the step executes, the status reflects the results of the execution. Possible status values can vary based on the type of step. Typical values include `Passed`, `Failed`, `Done`, and `Error`.

Test Program Editor (TSM)

Select **Semiconductor Module»Edit Test Program: <filename>** or click the **Edit Test Program: <filename>** button on the TSM toolbar to launch the Test Program Editor for the sequence file. Use this dialog box to specify the pin map and bin definitions files, create and edit test program configurations, and configure other settings for the test program.

The Test Program Editor contains the following panels:

- Pin Map—Specifies the path of the pin map file.
- Bin Definitions—Specifies the path of the bin definitions file.
- Digital Pattern Project—Specifies the path of the digital pattern project file.
- Offline Mode—Specifies the path of the Offline Mode system configuration file you want to associate with the test program.
- InstrumentStudio Project—Specifies the path of the InstrumentStudio project file.
- LabVIEW Project—Specifies the path of the LabVIEW project file.
- Specifications Files—Specifies a list of specifications file references available for use in the test program.
- PAT Algorithm Settings—Specifies values for the part average testing (PAT) algorithm settings for the test program.

- Test Limits Files—Specifies a list of test limits file references available for use in the test program. Each test program configuration can specify a test limits file to load when test program execution begins.
- Alarms— Specifies how alarms are handled on a per-alarm and per-pin basis.
- Configuration Definition—Specifies a table of standard and custom test conditions available for use in the test program. Each test program configuration can specify a value for the test conditions.
- Configurations—Specifies a list of configurations available for use in the test program.
- Configuration Panels—Specifies test condition values and test limits file for the configuration.

Test Program Editor (TSM)

Pin Map Panel

The Pin Map panel contains the following options:

- **Pin Map File Path**—Specifies the relative pathname to the pin map file to use in the test program.

  A red exclamation point indicates that an issue exists with the file, such as the file is invalid or that the file does not conform to the Pin Map XML schema. A tooltip displays the error message. If errors exist in the file, you make changes in the dialog box, and click **OK**, TSM commits those changes. Do not proceed without reviewing the errors for the pin map file.
- **Open file for edit** —Launches the pin map file in the Pin Map Editor.

See Also

Test Program Editor

Test Program Editor (TSM)

Bin Definitions Panel

The Bin Definitions panel contains the following options:

- **Bin Definitions File Path**—Specifies the pathname of the bin definitions file to use in the test program.

  A red exclamation point indicates that the file is invalid or that the file does not conform to the Bin Definitions XML schema. A tooltip displays the error message. If errors exist in the file, you make changes in the dialog box, and click **OK**, TSM commits those changes. Do not proceed without reviewing the errors in the bin definitions file.

- **Open file for edit** —Launches the bin definitions file in the Bin Definitions Editor.

See Also

Bin Definitions Editor

Test Program Editor

Test Program Editor (TSM)

Digital Pattern Project Panel

The Digital Pattern Project panel contains the following options:

- **Digital Pattern Project File Path**—Specifies the pathname of the digital pattern project file to use in the test program.

  A red exclamation point indicates that an issue exists for the file. A tooltip displays the error message. Do not proceed without reviewing the errors for the file.

- **Open file for edit** —Launches the digital pattern project file in the Digital Pattern Editor.

See Also

Test Program Editor

Test Program Editor (TSM)

Offline Mode Panel

The optional Offline Mode panel contains the following options:

- **Offline Mode System Configuration File Path**—Specifies the Offline Mode system configuration file you want to associate with the test program.

Test Program Editor (TSM)

InstrumentStudio Project Panel

The InstrumentStudio Project panel contains the following options:

- **InstrumentStudio Project File Path**—Specifies the pathname of the InstrumentStudio project file to use in the test program.

  A red exclamation point indicates that an issue exists for the file. A tooltip displays the error message. Do not proceed without reviewing the errors for the file.

- **Open file for edit** —Launches InstrumentStudio. If you specify an InstrumentStudio project file, this project opens in InstrumentStudio, otherwise the most recent project loads.

See Also

Test Program Editor

Test Program Editor (TSM)

LabVIEW Project Panel

Use the LabVIEW Project panel to link a sequence file to a LabVIEW project file, which allows the selected sequence file to populate pin, relay, relay configuration,

specification, published data ID, and input data ID controls in the specified LabVIEW project file. If you link multiple sequence files with different pin map or specification files, TSM combines the values into a single list when populating the controls. The LabVIEW Project panel contains the following options:

- **LabVIEW Project File Path**—Specifies the pathname of the LabVIEW project file to associate with the sequence file.
- **Open file for edit** —Launches the specified LabVIEW project file in LabVIEW.
- **Link/Unlink**—Links or removes the link from the selected LabVIEW project file to the sequence file currently open. The LabVIEW project file stores the sequence file link as a relative path. If the LabVIEW project file is currently open in LabVIEW, LabVIEW prompts you to save the project.

You can also link a LabVIEW project file to a sequence file in LabVIEW. Refer to **Linking a Sequence File to a LabVIEW Project File (TSM)** in the **LabVIEW Help** for more information.

See Also

Test Program Editor

Test Program Editor (TSM)

Specifications Files Panel

Click the **Add Specifications File** button or the **Remove Specifications File** button at the bottom of the panel to add or remove specifications files (`.specs`) to use with the test program. The Specifications Files panel contains the following options:

- **Specifications File Path**—Specifies the pathname of the specifications file to use in the test program.

  A red exclamation point indicates that the file is invalid or that the file does not conform to the Specifications XML schema. A tooltip displays the error

message. If errors exist in the file, you make changes in the dialog box, and click **OK**, TSM commits those changes. Do not proceed without reviewing the errors in the specifications file.

- **Open file for edit**  —Launches the specifications file in the application you associated with the `.specs` file extension, typically an XML editor.
- **Add Specifications File**—Adds a new specifications file reference to the list.
- **Remove Specifications File**—Removes the specifications file reference you select from the list.
- **Specifications Files in Digital Pattern Project list**—Displays the pathname for specifications files in the digital pattern project. A red exclamation point indicates that an issue exists with the file. A tooltip displays the error message.

See Also

Test Program Editor

Test Program Editor (TSM)

PAT Algorithm Settings Panel

If the test program settings do not match the PAT algorithm settings defined in the PAT callbacks sequence file, TSM displays one of the following buttons for you to take the appropriate action to ensure that the PAT algorithm settings match in both locations:

- **Remove Test Program Settings**—Removes the existing PAT algorithm settings from the test program if no PAT plug-in is installed or if the PAT callbacks sequence file does not include any PAT algorithm settings. If you remove the settings from the test program, you can add the settings back when you install a PAT plug-in that includes PAT algorithm settings.
- **Add Test Program Settings**—Adds the missing PAT algorithm settings to the test program when an installed PAT callbacks sequence file includes PAT algorithm settings.

- **Update Test Program Settings**—When applicable, updates the PAT algorithm settings in the test program to match the PAT algorithm settings defined in the installed PAT callbacks sequence file.

The Part Average Testing Algorithm Settings panel contains the following options when a PAT plug-in exists in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule` directory:

- **Name**—Specifies the display name of the PAT algorithm setting. If the PAT algorithm setting defines a description for the setting, a tooltip displays the description.
- **Value**—Specifies the value to use for the PAT algorithm setting in the test program.

  A red exclamation point indicates that an issue exists with the value, such as when a value is higher than the maximum value of the setting. A tooltip displays the error message. If errors exist in any value, you make changes in the dialog box, and click **OK**, TSM commits those changes. Do not proceed without reviewing the errors for the PAT algorithm settings.

See Also

Test Program Editor

Part Average Testing

Part Average Testing Algorithm Settings

Part Average Testing Examples

Test Program Editor (TSM)

Test Limits Files Panel

Click the **Add Test Limits File** button or the **Remove Test Limits File** button at the bottom of the panel to add or remove test limits file references available for use in the test program. Each test program configuration can specify a test limits file to

load when test program execution begins. The Test Limits Files panel contains the following options:

- **Test Limits File list**—Each entry in the list contains the following fields:
  - **Name**—Specifies a unique name for the test limits file reference. A red exclamation point indicates that the test limits reference name is invalid. A tooltip displays the error message.
  - **Test Limits File Path**—Specifies the relative pathname to the test limits file. A red exclamation point indicates that the file is invalid. A tooltip displays the error message.
  - **Open file for edit**  —Launches the test limits file in the default application associated with the file extension on the computer. This option is not available if the test limits file is embedded in the test program file.
- **Add Test Limits File**—Adds a new test limits file reference to the list.
- **Remove Test Limits File**—Removes the test limits file reference you select from the list.
- **Embed Test Limits File**—Embeds the contents of the selected test limits file in the test program sequence file. To protect limits files from modification or viewing, embed the limits file in the test program sequence file and use the TestStand password protection option to lock the sequence file. The Embed Test Limits File option is available when the test limits file path is valid and not already embedded in the sequence file.

Note NI does not recommend using passwords as the only way of protecting intellectual property.

- **Extract Test Limits File**—Extracts an embedded test limits file from the test program sequence file. Use this option to view or change the contents of the test limits file.

See Also

Test Program Editor

Exporting and Importing Test Limits with Text Files

Test Program Editor (TSM)

Alarms Panel

When you open the Alarms panel, TSM checks the pin map and queries each instrument driver to determine which alarms the instrument supports. The Alarms panel lists the pins connected to instruments with alarms support and the supported alarm types for each pin. Use this panel to enable alarms and specify the run-time behavior for each pin and alarm combination. When a test runs, TSM checks for alarms after each test step and executes the behavior you specified.

Note Enable Offline Mode to use alarms on a computer without access to NI instruments that have alarms support.

The Alarms panel contains the following options:

- **Alarms enabled**—Specifies whether alarms are enabled. Enable alarms if you want TSM to check for alarms after each step and execute the action specified in the **Behavior** column. Test times may increase when you enable alarms.
- **Pin**—Specifies the pin.
- **Alarm Type**—Specifies the type of alarm.

Note NI TestStand 2020 Semiconductor Module supports the `ComplianceAlarm` for NI-DCPower 20.1 and later. The `ComplianceAlarm` indicates that the instrument state was not at its programmed value or exceeded programmed limits when a measurement was made, which invalidates the measurement.

- **Behavior**—Specifies the action that TSM will execute if the alarm is raised. When you enable alarms and specify **Log**, **Fail step and any tests in step**, or **Error step and bin part to alarm bin** in the **Behavior** drop-down

menu, TSM displays a warning in the Output pane for each raised alarm. Select one of the following behaviors:

- **Ignore**—Ignore the alarm.
- **Log**—Log the alarm in the STDF Log file. TSM logs alarm information in the following STDF Log file records based on the test result data acquired:
  - Parametric Test Records (PTR) or Functional Test Records (FTR)—Logs the alarm type and associated pin in the ALARM_ID field.
  - Datalog Text Record (DTR)—Creates a new record for each alarm occurrence and displays the alarm type and associated pin.
  - Test Results Record (TSR)—Increments the alarm count stored in the ALRM_CNT field.
- **Fail step and any tests in step**—Set the step status to `Failed` and assign the default fail bin to the DUT.

Note If the test is part of a Semiconductor Multi Test step, all tests in the step are set to `Failed` and the fail bin you specified for the Semiconductor Multi Test step is assigned to the DUT. If no bin is specified, the default fail bin is assigned to the DUT.

- **Error step and bin part to alarm bin**—Set the step status to `Error` and generate a run-time error. If executing in the sequence editor, TSM will launch the Semiconductor Module Run-Time Error dialog box. The alarm bin you specified in the bin definitions file for the test program is assigned to the DUT.

Note If the test is part of a Semiconductor Multi Test step, all tests in the step are set to `Failed`.

See Also

Bin Definitions Editor

Test Program Editor

Test Program Editor (TSM)

Configuration Definition Panel

The Configuration Definition panel specifies a table of standard and custom test conditions available for use in the test program. Each test program configuration specifies a value for the test conditions.

The Configuration Definition panel contains the following options:

- **Test Condition Type**—Specifies the data type of the test condition. You can modify the type only or custom test conditions. When you modify a test condition type, each configuration resets the test condition value to a default value.
- **Test Condition Name**—Specifies the name of the test condition. Only custom test condition names can be modified. A red exclamation point indicates that the custom test condition name is invalid. A tooltip displays the error message.
- **Add Condition**—Launches the Specify New Condition Name dialog box, in which you can add a new test condition. You can select a standard test condition from a list or create a new custom test condition.
- **Delete Condition**—Removes the test condition you select from the test program and the test program configurations.

See Also

Test Program Editor

Test Program Editor (TSM)

Configurations Panel

The Configurations panel specifies a list of configurations available for use in the test program. When you specify a new configuration, the Test Program Editor updates with a new panel that matches the name of the configuration you added. Use the corresponding Configuration panel to specify test condition values and fields for configuring the test limits file for the configuration.

The Configurations panel contains the following options:

- **Configuration Name**—Specifies the name of the configuration. A red exclamation point indicates that the configuration name is invalid. A tooltip displays the error message.
- **Add Configuration**—Adds a configuration to the list.
- **Delete Configuration**—Removes the configuration you select from the list.

See Also

Test Program Editor

Test Program Editor (TSM)

Configuration Panels

Each configuration you specify uses a corresponding Configuration panel that contains a table to specify test condition values and fields for configuring the test limits file for the configuration.

The Configuration panels contains the following options:

- **Test Condition Name**—Displays the name of the test condition.
- **Test Condition Value**—Specifies the value for the test condition for the configuration.
- **Test Limits File**—Specifies the test limits file the configuration loads when test program execution begins.
- **Open for edit**  —Launches the test limits file in the default application associated with the file extension on the computer.
- **Test Limits File Import Mode**—Specifies whether the test limits file replaces only matching tests or deletes and adds new tests when loaded into the test program.

- **Require every step to be in test limits file**—Returns a run-time error and does not import the file if a test or step exists in the sequence but does not exist in the limits file. If you do not enable this option, the Import/Export Test Limits tool imports only matching tests or steps in the limits file.
- **Import into Sequence File**—Imports the test limits file into the sequence file.

See Also

Test Program Editor

Exporting and Importing Test Limits with Text Files

Bin Definitions Editor (TSM)

Use the Bin Definitions Editor to view, create, modify, and save bin definitions files instead of editing the XML files directly. Use the Bin Definitions panel in the Test Program Editor to specify a bin definitions file for a test program.

Select **Semiconductor Module»Edit Bin Definitions File** or click the **Edit Bin Definitions File** button on the TSM toolbar to launch the Bin Definitions Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Bin Definitions** in the Test Program Editor to launch the Bin Definitions panel. Click the **Open file for edit** button to launch the Bin Definitions Editor.

The Bin Definitions Editor validates the fields of the hardware and software tabs and uses a red error icon on the tab and in the corresponding fields to indicate errors to resolve, such as duplicate bin numbers, duplicate bin names, or software bins without assigned hardware bins. Hover over the error icon to display a tooltip with specific error information. You must resolve errors before you can save the file or close the editor using the **OK** button. The editor does not load files that are invalid according to the Bin Definitions schema.

When you close the editor, a dialog box prompts you to discard changes or return to the editor to save or cancel the changes to the bin definitions file if the file has been edited since you opened it in the editor.

Configuring Bin Definitions Files

The Bin Definitions Editor includes the following options and tabs:

- **Bin Definitions File Path**—Specifies the bin definitions file loaded.
- **Undo**—Removes the last edit made.
- **Redo**—Reinstates the last edit removed.
- **Open**—Launches the Select Bin Definitions File dialog box, in which you can browse to the bin definitions file to load.
- **Save**—Displays a context menu that includes a **Save** option to save the file to the current path and a **Save As** option to launch as Save As dialog box. The **Save** button is disabled when validation errors exist in the bin definitions file.
- **New**—Creates a new bin definitions file. If the file currently open in the editor has unsaved changes, a dialog box prompts you to discard the changes or return to the editor to save or cancel the changes to the file before creating a new file.
- **Hardware Bins tab**—Displays an editable table that defines a set of hardware bins. You can edit the cells directly or copy and paste rows of the table. The table includes the following sortable columns:
  - **Number**—Specifies the hardware bin number.
  - **Name**—Specifies the hardware bin name.
  - **Type**—Specifies an existing bin type. Options include Pass, Fail, and Other.
- **Software Bins tab**—Displays an editable table that defines a set of software bins. You can edit the cells directly or copy and paste rows of the table. The table includes the following sortable columns:
  - **Number**—Specifies the software bin number.
  - **Name**—Specifies the software bin name.
  - **Hardware Bin**—Specifies an existing hardware bin.
  - **Type**—Displays the existing hardware bin type. The specified hardware bin determines the value of this field.

- **Default Bins tab**—Displays the following options for the default bins to use when no bin is assigned to a DUT:
  - **Default Error**—The software bin to which TSM assigns a DUT when the main test sequence errors. The software bin must be associated with a hardware bin of the Fail or Other type.
  - **Default Pass**—The software bin to which TSM assigns a DUT when the main test sequence passes and the DUT has not yet been assigned to a bin. The software bin must be associated with a hardware bin of the Pass type.
  - **Default Fail**—The software bin to which TSM assigns a DUT when the main test sequence fails and the DUT has not yet been assigned to a bin. The software bin must be associated with a hardware bin of the Fail or Other type.
  - **Alarm**—The software bin to which TSM assigns a DUT when an alarm is raised and the behavior assigned to the alarm is **Error step and bin part to alarm bin**. You can assign the alarm behavior from the **Alarms** panel in the **Test Program Editor**.
- **Add Bin**—Adds a new hardware bin when on the Hardware Bins tab or a new software bin when on the Software Bins tab.
- **Delete Bin**—Deletes the selected bin or bins.
- **Software Bins Only**—Hides the Hardware Bins tab to specify only software bins, which can be helpful when a tester configuration does not support hardware bins, such as for wafer testing. Enabling this option clears the hardware bins. Use Undo to recover hardware bin information. Disable this option to return to normal editing mode. The editor saves this setting in the bin definitions file.

See Also

Bin Definitions Schema

Pin Map Editor (TSM)

Use the Pin Map Editor to view, create, modify, and save pin map files instead of editing the XML files directly. Use the Pin Map panel in the Test Program Editor to specify a pin map file for a test program. The pin map file also serves as the channel map file.

Select **Semiconductor Module»Edit Pin Map File** or click the **Edit Pin Map File** button on the TSM toolbar to launch the Pin Map Editor. Alternatively, you can select **Semiconductor Module»Edit Test Program** and then select **Pin Map** in the Test Program Editor to launch the Pin Map panel. Click the **Open file for edit** button to launch the Pin Map Editor.

The Pin Map Editor uses a red error icon in the Errors and Warnings window to indicate that the file does not conform to the Pin Map schema. Click the error icon or double-click the error message to highlight the error on the XML tab. The Errors and Warnings window displays a warning in orange text when the file conforms to the Pin Map schema but will generate an error at run time.

When you close the editor with the **Cancel** button and the pin map file has been edited since you opened it in the editor, a dialog box prompts you to discard changes or return to the editor. When you close the editor with the **OK** button, the file is updated on disk with the changes you made in the editor.

Configuring Pin Map Files

The Pin Map Editor includes the following options and tabs:

- **Pin Map File**—Specifies the pin map file to load. You can manually enter a relative file path.
- **Undo**—Removes the last edit made.
- **Redo**—Reinstates the last edit removed.
- **Open**—Launches the Select Pin Map File dialog box, in which you can browse to the pin map file to load.

- **Save**—Displays a context menu that includes a **Save** option to save the file to the current path and a **Save As** option to launch as Save As dialog box. The **Save** button is disabled when errors exist in the pin map file.
- **New**—Creates a new pin map file. If the file currently open in the editor has unsaved changes, a dialog box prompts you to discard the changes or return to the editor to save or cancel the changes to the file before creating a new file.
- **Pin Map tab**—Displays an editable, hierarchical view of the instruments, pins, pin groups, relays, relay groups, relay configurations, sites, and connections in the pin map file. For each item, you can edit the attributes of the item, insert new items, or insert comments in the file.
- **XML tab**—Displays the pin map file in a text format that you can edit.
- **Errors and Warnings window**—Displays issues to resolve. Click the **Goto Error in XML** error icon to highlight the error on the XML tab.

See Also

Common XML Validation Error Messages

Pin Map Schema

Pin Map Tab (TSM)

The Pin Map tab contains the following sections:

- **Instruments**—Use this section to specify the instruments required to execute the test program.
- **Pins**—Use this section to specify the DUT pins and other pin types connected to the tester.
- **Pin Groups**—Use this section to specify a grouping of pins that you can reference with a single name.
- **Relays**—Use this section to specify the relays on the tester that the test program associates with the pin map file references.

- Relay Groups—Use this section to specify a grouping of relays that you can reference with a single name.
- Relay Configurations—Use this section to specify a set of relays and their positions.
- Sites—Use this section to specify the sites on the tester.
- Connections—Use this section to specify mappings from pins to instrument channels for every site and for system resources.

Instruments (TSM)

Use the Instruments section on the Pin Map tab to specify the type of instruments required to execute the test program and the name and attributes of each instrument. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add an instrument to the pin map file:

- Click **<Add Instruments Here>** to display the Instruments pane, and click the button of the instrument type you want to add.
- Right–click **<Add Instruments Here>** and select the instrument type you want to add from the context menu.

The Pin Map Editor automatically adds the instrument to the **Instruments** section. Select an instrument in the **Instruments** section to display the Instruments pane, where you can edit the attributes of the instrument.

You can also cut, copy, and paste instruments, or add comments in the Instruments pane. Use the **Comment** button to specify a comment for the selected instrument. Comments display beneath the instrument they modify.

Notes

- Consider using the following instrument naming convention for semiconductor test programs: `InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation`, for

example, `HSD_657x_C2_S03`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), and `SlotLocation` uses double digits to identify the slot location (`Sxx`).

- Names for NI instruments in the pin map file are not case sensitive.

TSM supports the following instrument types and instrument attributes:

- **DCPower**—Defines an NI-DCPower instrument.
  - **Name**—Name of the instrument, as defined in Measurement & Automation Explorer (MAX).
  - **Number of Channels**—Number of channels available on the instrument.
  - **Channel Group Name/Channels(s) table**—Lists the channel groups and the channels assigned to each group. By default, the Pin Map Editor creates one channel group containing all instrument channels. Use the plus (+) or minus (-) buttons to add or remove channel groups.

Note A channel group is a collection of channels controlled by the same instrument session. NI-DCPower channel groups can contain channels from different physical NI-DCPower instruments.

  - **Channel Group Name**—Name of the channel group(s). The Channel Group Name is case sensitive and must not duplicate an instrument name or a group name on another instrument type.

Note NI-DCPower channels cannot be added to groups of other instruments.

  - **Channel(s)**—Channel(s) assigned to a channel group. When you add a new channel group, the Pin Map Editor prompts you to add channels to the new group. Define the channels as a comma-separated list (e.g.,

0,1,3,..,n), a continuous range (e.g., 0:3), or as a combination of the two (e.g., 0:1,3).

Note All channels for all instruments must be assigned to a channel group and no channel can assigned to more than one group.

- **Digital Pattern**—Defines an NI-Digital Pattern instrument.
  - **Name**—Name of the instrument, as defined in MAX.
  - **Number of Channels**—Number of channels available on the instrument.
  - **Group**—Name of the group that contains the instrument. By default, the Pin Map Editor sets this attribute to `Digital` when you add NI–Digital Pattern instruments to the pin map file. By using the same group name for all NI-Digital Pattern instruments, TSM combines all instruments into a single session so you can avoid session loops in code modules. To create multiple NI-Digital Pattern sessions, use a unique name for each set of instruments for which you want to create a session. Refer to the **Digital Pattern Help** for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

Note Instrument group names must be unique and must not duplicate instrument names in the pin map file.

- **RFSA**—Defines an NI-RFSA instrument. NI-RFSA instruments define a channel named `In`.
  - **Name**—Name of the instrument, as defined in MAX.
- **RFSG**—Defines an NI-RFSG instrument. NI-RFSG instruments define a channel named `Out`.
  - **Name**—Name of the instrument, as defined in MAX.
- **VST**—Defines an NI-VST instrument that can hold RFSA, RFSG, and FPGA sessions. NI-VST instruments define a channel named `In` and another channel named `Out`.

- **Name**—Name of the instrument, as defined in MAX.
- **Custom FPGA File**—(Optional) path to the FPGA file relative to the path of the pin map file. You can manually specify an absolute file path.

- **RFPM**—Defines an RF Port Module instrument that can hold RFPM, RFmx, RFSA, RFSG, and FPGA sessions.
  - **Name**—Name of the VST instrument, as defined in MAX, that is part of the RF port module subsystem.
  - **Custom FPGA File**—(Optional) path to the FPGA file relative to the path of the pin map file. You can manually specify an absolute file path.
  - **Calibration File**—Path, relative to the path of the pin map file, to the TDMS files that contain the calibration data for the RF Port Module instrument. You can manually specify an absolute file path.
  - **IVI Switch Resource Name**—IVI Switch resource name associated with the port module, as defined in MAX.
  - **Ports List**—Defines the ports available in the RF Port Module in a comma-separated list of numbers or ranges of numbers separated by a hyphen. Port number ranges are inclusive and must be in ascending order, for example, `channelList="2,3,4-8"`.
- **HSDIO**—Defines an NI-HSDIO instrument.
  - **Name**—Name of the instrument, as defined in MAX.
  - **Number of Channels**—Number of channels available on the instrument.
  - **PFI Lines**—(Optional) Defines the PFI lines available in the NI-HSDIO instrument in a comma-separated list of numbers or ranges of numbers separated by a hyphen. PFI number ranges are inclusive and must be in ascending order, for example, `PFILines="2,3,4-8"`.
- **DMM**—Defines an NI-DMM instrument. NI-DMM instruments define a single channel, displayed within TSM as channel `0`.
  - **Name**—Name of the instrument, as defined in MAX.

- **SCOPE**—Defines an NI-SCOPE instrument.
  - **Name**—Name of the instrument, as defined in MAX.
  - **Number of Channels**—Number of channels available on the instrument.
  - **Group**—Name of the group that contains the instrument. By default, the Pin Map Editor sets this attribute to `Scope` when you add NI–SCOPE instruments to the pin map file. By using the same group name for all NI-SCOPE instruments, TSM combines all instruments into a single session so you can avoid session loops in code modules. To create multiple NI-SCOPE sessions, use a unique name for each set of instruments for which you want to create a session. Refer to the **NI-SCOPE Help** for information about hardware limitations that prevent certain instruments from operating together as a single instrument.

Note Instrument group names must be unique and must not duplicate instrument names in the pin map file.

- **FGEN**—Defines an NI-FGEN instrument.
  - **Name**—Name of the instrument, as defined in MAX.
  - **Number of Channels**—Number of channels available on the instrument.
- **DAQmx**—Defines an NI-DAQmx task, not an instrument.
  - **Name**—Name of the task, as defined in test program code modules.
  - **Task Type**—Category of the task. Pin queries that return tasks of more than one task type return an error.
  - **Channel List**—List of physical channels associated with the task.
- **Relay Driver**—Defines a PXI-2567 relay driver module.
  - **Name**—Name of the relay driver module, as defined in MAX.
  - **Number of Control Lines**—Number of control lines available on the relay driver module.

- **Multiplexer**—Defines a switching instrument to use as a multiplexer across multiple test sites. You can use one instrument multiplexed across multiple test sites or multiple instruments multiplexed across multiple test sites.
  - **Name**—Name of the Switch Executive virtual device, as defined in MAX.
  - **Multiplexer Type**—(Optional) String that identifies the switch type, family, class, or product group. You cannot specify a value that begins with `ni`. This value is a string that you define in the pin map and is not a predefined value from some other source, such as a name in MAX, that you select. Use this value to identify all instances of a particular switch type. Switches of the same type typically have the same session data type and same driver API.
- **Custom Instrument**—Defines an instrument that TSM does not natively support. Use the TSM Code Module API to set any type of session data on a channel, group of channels, or instrument. Refer to the `<TestStand Public>\Examples\NI_SemiconductorModule\Custom Instruments` directory for examples of using TSM pin map files and VIs to perform tests using instruments that TSM does not natively support.
  - **Name**—Identifies the instrument. For instruments that NI provides but that TSM does not natively support, specify the name of the instrument, as defined in MAX.

    Note Names for custom instruments in the pin map file are case sensitive.
  - **Instrument Type Id**—String that identifies the instrument type, family, class, or product group. You cannot specify a value that begins with `ni`. This value is a string that you define in the pin map and is not a predefined value from some other source, such as a name in MAX, that you select. Use this value to identify all instances of a particular instrument type. Instruments of the same type typically have the same session data type and same driver API.
- **Model-Based Instrument**—Defines a Model-Based Instrument.
  - **Name**—Unique string that identifies the instance of the Model-Based Instrument in the pin map.

- **Instrument Model**—List of installed model description files in the Instrument Model Library.
- **Category**—Category of the ModelBasedInstrument. The category is set in the instrument model and cannot be changed.
- **Subcategory**—Subcategory of the ModelBasedInstrument. The subcategory is set in the instrument model and cannot be changed.
- **Property/Value Tables**—Editable tables of instrument and resource properties as defined in the model description file. The first table contains properties for the entire instrument. Subsequent tables contain properties for specific resources of the instrument.

Pins (TSM)

Use the **Pins** section on the Pin Map tab to specify the DUT pins and other pin types connected to the tester. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add a pin connection to the pin map file:

- Click **<Add Pins Here>** to display the Pins pane, and click the button of the pin type you want to add.
- Right–click **<Add Pins Here>** and select the pin type you want to add from the context menu.

The Pin Map Editor automatically adds the pin connection to the **Pins** section. Select a pin connection in the **Pins** section to display the Pins pane, where you can edit the pin's name.

Note Pin names are case sensitive, must begin with a letter or underscore (_), and are limited to A-Z, a-z, 0-9, or _ characters.

You can also cut, copy, and paste pins, or add comments in the Pins pane. Use the **Comment** button to specify a comment for the selected pin. Comments display beneath the pin they modify.

TSM supports the following pin connection types:

- **DUT pin**—Specifies a DUT pin, which is a pin on a DUT or a resource on the tester or DIB that is associated with one or more sites.
- **System pin**—Specifies a system pin, which is resource on the tester or DIB that is connected to an instrument

---

See Also

Connecting Shared Resources in the Pin Map

Pin Groups (TSM)

Use the **Pin Groups** section on the Pin Map tab to specify a grouping of pins that you can reference with a single name.

Choose one of the following options to add a pin group to the pin map file:

- Click **<Add Pin Groups Here>** to display the Pin Groups pane, and click the **Pin Group** button.
- Right–click **<Add Pin Groups Here>** and select Pin Group from the context menu.

The Pin Map Editor automatically adds the pin group to the **Pin Groups** section. Select a pin group in the **Pin Groups** section to display the Pin Groups pane, where you can change the pin group name and use the individual checkboxes or the **Select All** checkbox to add or remove DUT pins, system pins, or pin groups from the selected pin group.

Note Pin group names are case sensitive, must begin with a letter or underscore (_), and are limited to A-Z, a-z, 0-9, or _ characters.

Once you create a pin group, you can also add pins or other pin groups to the pin group using one of the following options:

- Click **<Add Pin References Here>**, and click the **Pin Reference** button in the Pin Groups pane to display a drop–down menu of available pins and pin groups.
- Right–click **<Add Pin References Here>**, and select Pin Reference from the context menu to display a drop–down menu of available pins and pin groups.
- Drag pins or pins groups from the **Pins** or **Pin Groups** section into a pin group.

You can also cut, copy, and paste pins, or add comments in the Pin Groups pane. Use the **Comment** button to specify a comment for the selected pin group. Comments display beneath the pin group they modify.

Relays (TSM)

Use the **Relays** section on the Pin Map tab to specify the relays on the site and the relays on the tester that the test program associates with the pin map file references. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add a relay to the pin map file:

- Click **<Add Relays Here>** to display the Relays pane, and click the button of the relay type you want to add.
- Right–click **<Add Relays Here>** and select the relay type you want to add from the context menu.

The Pin Map Editor automatically adds the relay to the **Relays** section. Select a relay in the **Relays** section to display the Relays pane, where you can edit the relay's name, open state display label, and closed state display label.

Note Relay names are case sensitive, must begin with a letter or underscore (_), and are limited to A-Z, a-z, 0-9, or _ characters.

You can also cut, copy, and paste relays, or add comments in the Relays pane. Use the **Comment** button to specify a comment for the selected relay. Comments display beneath the relay they modify.

TSM supports the following relay types:

- **Site relay**—Specifies a site relay, which is a relay on the tester or DIB that is connected to a relay driver module and that is associated with one or more sites.
- **System relay**—Specifies a system relay, which is a relay on the tester or DIB that is connected to a relay driver module and that is associated with all sites.

See Also

Connecting Shared Resources in the Pin Map

Relay Groups (TSM)

Use the **Relay Groups** section on the Pin Map tab to specify a grouping of relays that you can reference with a single name. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add a relay group to the pin map file:

- Click **<Add Relay Groups Here>** to display the Relay Groups pane, and click the **Relay Group** button.
- Right–click **<Add Relay Groups Here>** and select Relay Group from the context menu.

The Pin Map Editor automatically adds the relay group to the **Relay Groups** section. Select a relay group in the **Relay Groups** section to display the Relay Groups pane, where you can change the relay group name and use the individual checkboxes or the **Select All** checkbox to add or remove site relays, system relays, or relay groups from the selected relay group.

Note Relay group names are case sensitive, must begin with a letter or underscore (_), and are limited to A-Z, a-z, 0-9, or _ characters.

Once you create a relay group, you can also add relays or other relay groups to the relay group using one of the following options:

- Click **<Add Relay References Here>**, and click the **Relay Reference** button in the Relay Groups pane to display a drop–down menu of available relays and relay groups.
- Right–click **<Add Relay References Here>**, and select Relay Reference from the context menu to display a drop–down menu of available relays and relay groups.
- Drag relays or relay groups from the **Relays** or **Relay Groups** section into a relay group.

You can also cut, copy, and paste relay groups, or add comments in the Relay Groups pane. Use the **Comment** button to specify a comment for the selected relay group. Comments display beneath the relay group they modify.

Relay Configurations (TSM)

Use the **Relay Configurations** section on the Pin Map tab to specify a set of relays and their relay position states. You can set the relay state position of all relays with a single call to the TSM Code Module API. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add a relay configuration to the pin map file:

- Click **<Add Relay Configurations Here>** to display the Relay Configurations pane, and click the **Relay Configuration** button.
- Right–click **<Add Relay Configurations Here>** and select Relay Configuration from the context menu.

The Pin Map Editor automatically adds the relay configuration to the **Relay Configurations** section. Select a relay configuration in the **Relay Configurations**

section to display the Relay Configuration pane, where you can change the relay configuration name, select the relay position state, view the final relay state, and change the order of relays and relay groups. Relay state positions are applied sequentially in the order in which the relay or relay group appears in the Relay Configuration pane. You can change the order of a relay or relay group by using the **Up** and **Down** buttons to move the selected relay or relay group.

Alternatively, when you create a new relay configuration, you can click **<Add Relay Positions Here>**, then click the **Relay Position** button. Select the relay or relay group you want to add from the top drop–down menu, then select the relay position state from the bottom drop–down menu. Use the Relay Configuration pane to make additional changes.

Click the **Relay Configurations** section to display the Relay Configurations table, which includes a column for each relay and a column for each relay configuration. Each relay configuration column shows the final relay state of the corresponding relay(s).

You can also cut, copy, and paste relay configurations, or add comments in the Relay Configurations pane. Use the **Comment** button to specify a comment for the selected relay configuration. Comments display beneath the relay configuration they modify.

Sites (TSM)

Use the **Sites** section on the Pin Map tab to specify the sites on the tester. Site numbers start at `0` and must be consecutive without gaps. You can right–click any item in the section you want to edit and use the context menu to complete common tasks.

Choose one of the following options to add a site to the pin map file:

- Click **<Add Sites Here>** to display the Sites pane, and click the **Site** button.
- Right–click **<Add Sites Here>** and select Site from the context menu.

The Pin Map Editor automatically adds the site to the **Sites** section. Select a site in the **Sites** section to display the Sites pane.

You can also cut, copy, and paste sites, or add comments in the Sites pane. Use the **Comment** button to specify a comment for the selected site. Comments display beneath the site they modify.

Adding a Connection in the Pin Map Editor (TSM)

Use the **Connections** section on the Pin Map tab to specify mappings among pins, relays, sites, instruments, instrument channels, relay driver modules, and relay driver control lines.

1. Navigate to **Semiconductor Module»Edit Pin Map File** to launch the Pin Map Editor.
2. In the Pin Map tab, click **<Add Connections Here>** to open the Connections pane.
3. Click the connection type you want to add. Select from the options below.

Connection Type	Description
Connection	A connection between a DUT pin and an instrument channel for one or more sites
System Connection	A direct connection between a system pin and an instrument channel
Multiplexed Connection	A multiplexed connection between the same DUT pin on multiple sites and a single instrument channel
Relay Connection	A connection between a site relay and a relay driver module control line for one or more sites
System Relay Connection	A direct connection between a system relay and a control line of a relay drive module

4. Configure the connection.
   The Pin Map Editor automatically adds the connection to the **Connections** section.

5. (Optional) Click a connection to view and edit it in the Connections pane. You can also view and edit all connections in the Connections table.
6. (Optional) Add a comment to help keep connections organized.
    a. Click on the connection where you want to add a comment.
    b. In the Connections pane, click **Comment**.
    c. Add your comment in the **Comment** field.
    TSM displays the comment above the connection.

Viewing and Editing Connections in the Connections Table (TSM)

Use the Connections table to view and edit connections and connection attributes. The Connections table includes a row for every connection that can be made with the available pins, relays, and sites.

1. Navigate to **Semiconductor Module»Edit Pin Map File** to launch the Pin Map Editor.
2. In the Pin Map tab, click **Connections** to open the Connections table.
3. (Optional) Use the **View Connections for** drop-down menu to filter the table using one of the following categories.
    - **All Pins and Relays**
    - **DUT Pins and Site Relays by Site**
    - **System Pins and Relays**
4. (Optional) Click a column header to sort the column so you can navigate values faster.
5. Edit the values in the table directly or by using the drop-down menus. Refer to the descriptions below to understand the columns based on whether they describe a pin or relay.

Column Name	Description
Pin	For pins, the name of a device pin you connect to an instrument channel

	For relays, the name of a relay you connect to a control line on a relay driver module
Site	The test site or sites of the connection. After selecting an instrument or relay driver module in the **Instrument** column, enter a comma-separated list of site numbers to allow multiple sites to share the connection.
Instrument	For pins, the name of the instrument to connect For relays, the name of the relay driver module to connect Select **<Disconnect>** to remove the association between the pin and the instrument or the relay and the relay driver module.
Channel	For pins, an instrument channel number to assign to the pin For relays, a control line on a relay driver module to assign to the relay
Multiplexer	If applicable, specifies the multiplexer required to create the route
Route	If applicable, specifies the multiplexer route required to connect the pin and site to the instrument and channel

TSM Step Types

The TSM folder in the Step Types list on the Insertion Palette pane in the TestStand Sequence Editor contains the following step types:

- Semiconductor Multi Test—Evaluates one or more parametric or functional tests for the DUT. A single Semiconductor Multi Test step can specify multiple parametric or functional tests. You can configure multisite and binning options directly on the step.
- Semiconductor Action—Performs an action, such as instrument configuration, with access to the pin map and per-site inputs. You can configure multisite and per-site input options directly on the step.

- Semiconductor Sequence Call Step—Calls a sequence and propagates tests to Semiconductor Multi Test steps in the called sequence.
- Inline QA Test Block—Inserts a block of Inline QA Test Block and End steps, in which you can insert additional steps that call code modules that perform the inline QA tests.
- Set and Lock Bin—Uses an expression to assign a software bin to a DUT and overrides TSM automatic bin assignment.
- Set Relays—Controls relays and applies relay configurations.
- Get Test Information Step —Obtains the values for lot settings, station settings, STS state, execution data, and custom test conditions.
- Control STS Test Head Step—Controls properties of the STS.
- Load Correlation Offsets Step—Loads and applies correlation offset values to test results on a per-site basis at run time before evaluating the test result data against limits.
- Perform Part Average Testing Step—Performs part average testing for any tests with part average testing enabled that have already been performed for the current part.
- RF Steps—A series of TestStand step types you use to create measurement sequences for RF instruments. For more information, refer to the RF Steps documentation. You must have the STS Software Bundle installed to use RF Steps and the RF Steps documentation.

Note TSM steps disable the Switching panel of the Properties tab in the Step Settings pane. Use relays in the pin map to perform switching operations. To use the NI Switch Executive to perform switching operations, use the NI Switch Executive API in code modules.

TSM Step Templates

TSM provides a mechanism for other add-ons to install step type templates, such as TSM Mixed Signal Steps. Use the step types with template code to perform common operations, such as setting up and closing instruments, powering up a DUT, or executing common tests. You can modify the code to customize the behavior of the step within a test program.

Adding Step Templates to a Test Program

When you add a step template to a test program, the step prompts you to specify a location for the corresponding predefined VI or .NET assembly the step generates, based on the current module adapter setting. The step executes the corresponding code module from the location you specify.

Notes

- Some of the generated template VIs are not compatible with all the versions of LabVIEW that TSM supports. TSM generates a template VI that is compatible with the version of LabVIEW that has most recently been run on the computer.
- If you use the LabVIEW Adapter, you must have the LabVIEW Development System installed, and you must configure the LabVIEW Adapter to use the LabVIEW Development System.
- If you use the .NET Adapter, TSM generates the template code for all the step type templates in a single .NET assembly when you add any of the step type templates to the test program. Additional step type templates you add to the test program automatically use the existing instance of the .NET assembly.

Configuring the Step

The template code module for each step type uses default values for most input parameters. You must configure settings, such as limits and the pin(s) to use for the operation, for any tests associated with the step.

Step types that perform tests are based on the Semiconductor Multi Test step type. Step types that do not perform tests are based on the Semiconductor Action step type.

Modifying the Code Templates

Modifying the code module template for a step type affects the code module TSM generates when you add a new instance of the step type to any test program on the computer.

TSM stores LabVIEW code module templates in directories based on the earliest version of LabVIEW the template supports. To modify a LabVIEW code module template, copy the template VI directory from one of the LabVIEW version-specific directories in the <TestStand>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates directory to the <TestStand Public>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates directory within the corresponding LabVIEW version-specific directory and make changes to the template VIs in that location. If you want to use subVIs in the LabVIEW code module template, store the subVIs in a SubVIs directory within the template directory. Use the following syntax for the subVI filename: NameOfTemplate - SubVIName.vi.

For example, to modify the Continuity Test template for LabVIEW 2018 and later, copy the <TestStand>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\LabVIEW\LabVIEW 2018\Continuity Test\Template - Continuity Test.vi to the <TestStand Public>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\LabVIEW 2018 directory and copy all the subVIs with the prefix Continuity Test – from the <TestStand>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\LabVIEW 2018\Continuity Test\SubVIs directory to the <TestStand Public>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\LabVIEW 2018\Continuity Test\SubVIs directory.

To create LabVIEW template VIs to use in specific versions of LabVIEW, copy the customized VIs to directories that correspond to the year-based version of LabVIEW you want to use. For example, to use the modified Continuity Test LabVIEW template when LabVIEW 2019 or later is active, copy it to a <TestStand Public>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\LabVIEW 2019 directory. TSM uses this version of the Continuity Test template when

LabVIEW 2019 is active, even if a version of the Continuity Test template also exists in the LabVIEW 2018 directory.

To modify a .NET code module template, copy the <TestStand>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\DotNet directory to the <TestStand Public>\Components\Modules\NI_SemiconductorModule\StepTypeTemplates\DotNet directory, make changes to the .NET source in this location, and rebuild the .NET assembly to update the template. After you rebuild the .NET assembly, rename it to use the Template – prefix so the step types detect it properly.

Semiconductor Multi Test Step

Use the Semiconductor Multi Test step to evaluate one or more parametric or functional tests for the DUT. A single Semiconductor Multi Test step can specify multiple parametric or functional tests. You can configure multisite, binning, and per-site input options directly on the step.

Note You cannot use multiple Semiconductor Multi Test steps configured to use multiple threads in While loops, in Do While loops or in For loops that use the Custom Loop option when performing multisite testing. The Semiconductor Multi Test step reports a run-time error in these situations. Use other types of loops instead, such as For loops that use the Fixed Number of Iterations option.

Configuring the Step

You can configure a single Semiconductor Multi Test step test to evaluate a Boolean or numeric measurement against specified limits. Each numeric limit test can have independent limits, base units, comparison type, and software bin. Configure each measurement the same way you configure an individual Numeric Limit Test step. A Semiconductor Multi Test step passes when none of the specified tests fail.

Use the Semiconductor Multi Test step edit tabs in the TestStand Sequence Editor to specify the tests, comparison, limits, multisite, binning, and per-site input options.

Step Properties

The Semiconductor Multi Test step type defines a set of step properties and constants, in addition to the built-in properties common to all TestStand steps.

See Also

Test Program Editor

Exporting and Importing Test Limits with Text Files

Multisite Programming Techniques

Semiconductor Multi Test Step Execution Overview

The Semiconductor Multi Test step uses the following process when executing:

1. Creates a `SemiconductorModuleContext` object and stores it on the Step.SemiconductorModuleContext property if the site that executes the step is the working site. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system.
2. Replaces each test that specifies a pin group with a set of tests that are equivalent to the pin group test and deletes the pin group test, except for the following differences:
   - The Pin is set to the name of the pin in the pin group.
   - The Test Number is computed by adding the test number specified for the pin group test to the zero-based index of the pin in the pin group.
   - The Test Name is computed by appending the name of the pin in the pin group to the test name specified for the pin group test.

   Note This replacement happens the first time the step executes.
3. Calls a code module that completes the following actions:

a. Receives the `Step.SemiconductorModuleContext` property as an input parameter.
b. Obtains instrument channel and session information using the TSM Code Module API.
c. Performs measurements.
d. Publishes measurement data using the TSM Code Module API.

4. Performs tests. For each test, the step completes the following actions:
   a. Obtains the measurement value from the published measurement data or from the expression, if specified, in the **Test Data Source** column on the Tests tab.
   b. Adds the correlation offset specified in the currently loaded correlation offsets file, if any, to the measurement value.
   c. Compares the measurement value against the specified limits.
   d. Sets the test status depending on the evaluation type, limits, or measurement value, as shown in the following table.

Evaluation Type	Condition	Status
Numeric Limit	You specify a low limit and high limit	`Passed` or `Failed`
	You do not specify a low li mit and high limit	`Done`
Pass/Fail	The measurement value is `True`	`Passed`
	The measurement value is `False`	`Failed`
None	None	`Done`

Note When you enable the **Stop Performing Tests after First Failure** option on the Options tab and a test fails, TSM sets the status of the remaining Numeric Limit and Pass/Fail tests to `Skipped`. The step sets the status for tests

with the **None** evaluation type to `Done` regardless of whether any test fails.

e. Stores the measurement value in the expression, if specified, in the **Export Data To** column on the Tests tab.

f. Assigns a software bin to the DUT if the test fails and a bin has not yet been assigned to the DUT. If you have already assigned a bin using the Set and Lock Bin step and the test fails, the step reports a run-time error.

g. Continues or stops performing tests depending on whether a test failed and whether you enable the **Stop Performing Tests after First Failure** option on the Options tab. The step stores the measurement value in the Export Data To expression for tests with the **None** evaluation type, regardless of whether a test failed.

5. Sets the step status to `Passed` if all tests pass. Otherwise, sets the step status to `Failed` or `Error`.

Semiconductor Multi Test Step Properties

The Semiconductor Multi Test step type defines the following step properties:

- **Result.Evaluations**—An array that stores the tests you configure for the step on the Tests edit tab. The NI_SemiconductorModule_Evaluation data type defines the following fields:
  - **EvaluationType**—The type of evaluation the test performs. The default evaluation type is Numeric Limit.
  - **Pin**—A string that stores the test pin or pin group.
  - **MeasurementId**—A string that stores the published data ID.

Note The TSM Code Module API uses the `EvaluationType`, `Pin`, and `MeasurementId` properties to publish measurements from the code module to the corresponding tests the Semiconductor Multi Test step specifies.

- **TestNumberExpr**—An expression that determines the test number at run time. If this expression is not empty, the Semiconductor Multi Test step evaluates the expression and copies the evaluated value to the `TestNumber` property.
- **TestNumber**—A number that stores the test number.
- **FailBinExpr**—An expression that determines the software fail bin for the test at run time. The expression must evaluate to a valid software bin number. If the test fails and this expression is not empty, the Semiconductor Multi Test step evaluates the expression and copies the evaluated value to the `FailBin` property.
- **FailBin**—A number that stores the software fail bin for the test.
- **TestNameExpr**—An expression that determines the test name at run time. If this expression is not empty, the Semiconductor Multi Test step evaluates the expression and copies the evaluated value to the `TestName` property.
- **TestName**—A string that stores the test name.
- **MeasurementSourceExpr**—An expression that determines the numeric or Boolean measurement value at run time. If this expression is empty, the step obtains the measurement data from the TSM Code Module API in the code module the step calls.
- **MeasurementDestinationExpr**—Optional. An expression that specifies a custom location to store the numeric or Boolean measurement value after the step performs the test.
- **Status**—A string that stores the `Passed`, `Failed`, `Done`, `Skipped`, or `Error` test result.
- **NumericLimit**—The NI_SemiconductorModule_Evaluation_NumericLimit data type defines the following fields:
    - **Data**—A number that stores the numeric measurement value. The step obtains this value from the TSM Code Module API in the code module the step calls or from the `MeasurementSourceExpr` property.

- LowLimitExpr, **LowLimit**, **HighLimitExpr**, and **HighLimit**—The limits for the test.
- ComparisonType—The type of comparison, such as `"GELE"`.
- Units—A string that stores the base units for the limits and measurement value.
- **DataScalingExponent, LowLimitScalingExponent, HighLimitScalingExponent**—Numbers that determine the scaling factors for the measurement value and each limit value. Each of these properties contains an integer value that TSM uses as an exponent for the number 10 to determine the scaling factor.
- **CorrelationOffset**—The correlation offset value to apply to test results on a per-site basis at run time before evaluating the test result data against limits. The Load Correlation Offsets step sets this property.
- **SimulatedData**—The SimulatedData container uses the following fields to specify values for tests in Offline Mode:
  - **SimulatedDataUsageType**—Set to **AllSites** to specify a single value for all sites. Set to **PerSite** to specify specific values for each site. Set to **UsePublishedValue** if you want to use the published values. The default value is **None**, which specifies the default behavior of Offline Mode to ignore the values of **AllSite**, **PerSite**, and the published values.
    - **AllSites**—Specifies a single value for all sites.
    - **PerSite**—Specifies specific values for each site.

- **PassFail**—The NI_SemiconductorModule_Evaluation_PassFail data type defines the following fields:
  - Data—A Boolean value the step obtains from the TSM Code Module API in the code module the step calls or from the `MeasurementSourceExpr` property.
  - **SimulatedData**—The SimulatedData container uses the following fields to specify values for tests in Offline Mode:

- **SimulatedDataUsageType**—Set to **AllSites** to specify a single value for all sites. Set to **PerSite** to specify specific values for each site. Set to **UsePublishedValue** if you want to use the published values. The default value is **None**, which specifies the default behavior of Offline Mode to ignore the values of **AllSite**, **PerSite**, and the published values.
  - **AllSites**—Specifies a single value for all sites.
  - **PerSite**—Specifies specific values for each site.

- **Result.SemiconductorCommon**—A container that stores run-time data about the step. The NI_SemiconductorModule_SemiconductorCommon data type defines the following field:
  - **Sites**—A string that stores a comma-separated list of sites tested in the current thread, as shown in the Multisite Execution Diagram of the step. The step sets this property at run time.
  - **RaisedAlarms**—A string that stores a comma-separated list of alarms that were raised by the step. The step sets this property at run time.
- **SemiconductorModuleContext**—An object reference that stores the `SemiconductorModuleContext` object the step code module uses. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system. Code modules use this property to access the TSM Code Module API.
- **Multisite**—A container that stores the multisite options you configure for the step on the Options edit tab. The NI_SemiconductorModule_Multisite data type defines the following fields:
  - **SynchronizationOption**—The multisite option. The default option is **One thread per subsystem**.
  - **SpecifyPinsAndRelaysOption***—The option for how you specify what pins and relays to include in the `SemiconductorModuleContext` object for the step.
  - **PinsAndRelaysExpression***— The expression that specifies what pins and relays to include in the `SemiconductorModuleContext` object for

the step. The step uses this expression only when the `SpecifyPinsAndRelaysOption` property is set to 1.

- **DUTPinFilterOption***—The option for determining what pins to include in the `SemiconductorModuleContext` object for the step. By default, all DUT pins are included. If you try to access a pin that is not included in the `SemiconductorModuleContext` object, the step returns an error.
- **IncludedDUTPins***—An array that stores the DUT pins to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `DUTPinFilterOption` property is set to 1.
- **ExcludedDUTPins***—An array that stores the DUT pins to exclude from the `SemiconductorModuleContext` object for the step. The step uses this array only when the `DUTPinFilterOption` property is set to 0.
- **IncludeSystemPins***—A Boolean value that specifies whether to include all system pins in the `SemiconductorModuleContext` object for the step.
- **IncludedPinGroups***—An array that stores the pin groups to include in the `SemiconductorModuleContext` object for the step.
- **SpecifySiteRelays***—The option to manually specify which site relays to include in the `SemiconductorModuleContext` object for the step. By default, no site relays are included.
- **IncludedSiteRelays***—An array that stores the site relays to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.
- **IncludedRelayGroups***—An array that stores the relay groups to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.
- **IncludedRelayConfigurations***—An array that stores the relay configurations to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.

- **IncludeSystemRelays***—A Boolean value that specifies whether to include all system relays in the `SemiconductorModuleContext` object for the step.

- **EvaluationFailureOption**—The test failure option. The default value is stop on failure.
- **DotNetRuntimeData**—A hidden object reference that substeps of the Semiconductor Multi Test step use to cache run-time information about the step. Do not directly interact with this property.

Notes

- The numeric representation for the numeric limit data and limit values must be double-precision, 64-bit floating-point values, and cannot be signed or unsigned 64-bit integers.
- The `TestNameExpr`, `TestNumberExpr`, and `FailBinExpr` properties do not correspond to options on the Semiconductor Multi Test step edit tabs.
- * The Semiconductor Multi Test step uses these properties only when the **SynchronizationOption** is set to **One thread per subsystem** and the step is executing in a test socket thread. Steps in process model callback sequences such as `ProcessSetup` and `ProcessCleanup` include all DUT pins and system pins, and all site and system relays.

Semiconductor Multi Test Step Constants

Use the following constants with the Semiconductor Multi Test step properties to configure options for the step:

- **EvaluationType**—Use the following constants with the Step.Result.Evaluations[i].EvaluationType property:
  - 0—Numeric Limit test
  - 1—Pass/Fail test
  - 2—None evaluation
- **ComparisonType**—Use the following constants with the Step.Result.Evaluations[i].NumericLimit.ComparisonType property:
  - "GELE"—Low <= X <= High
  - "GTLT"—Low < X < High
  - "GELT"—Low <= X < High
  - "GTLE"—Low < X <= High
  - "LOG"—No comparison
- **SynchronizationOption**—Use the following constants with the Step.Multisite.SynchronizationOption property:
  - 0—One thread per site
  - 1—One thread only
  - 2—One thread per subsystem
- **SpecifyPinsAndRelaysOption**—Use the following constants with the Step.Multisite.SpecifyPinsAndRelaysOption property:
  - 0—Specify manually
  - 1—Use expression
- **DUTPinFilterOption**—Use the following constants with the Step.Multisite.DUTPinFilterOption property:
  - 0—Include all DUT pins except those listed in the ExcludedDUTPins array.
  - 1—Exclude all DUT pins except those listed in the IncludedDUTPins array.
- **SpecifySiteRelays**—Use the following constants with the Step.Multisite.SpecifySiteRelays property:

- `False`—Include all site relays.
- `True`—Include only the site relays listed in the IncludedSiteRelays array.

- **TestFailureOption**—Use the following constants with the Step.EvaluationFailureOption property:
  - `0`—Stop on failure. The step stops performing tests after the first test fails.
  - `1`—Continue on failure. The step performs all tests even if one test fails.

Semiconductor Multi Test Edit Tabs

Use the Semiconductor Multi Test step edit tabs in the TestStand Sequence Editor to specify the tests, limits, multisite, and binning options for the Semiconductor Multi Test step.

The Step Settings pane for the Semiconductor Multi Test step contains the following tabs:

- Tests—Configure the tests for the step. Each test specifies the test number, test name, pin or pin group, published data ID, limits, scaling factor, base units, software bin, evaluation type, test data source, and measurement destination.
- Per-Site Inputs—Configure the per-site inputs for the step. Each per-site input specifies a data value that you access in a code module using the TSM Code Module API.
- Options—Configure multisite and other options for the step.
- Part Average Testing—Configure part average testing (PAT) on the tests for the step.

See Also

Test Program Editor

Exporting and Importing Test Limits with Text Files

Tests Tab

The Tests tab contains a table that lists the tests the step performs. The following step types display a Tests tab:

- Semiconductor Multi Test step
- Semiconductor Sequence Call step
- Perform Part Average Testing step

Changing the Layout and Entering Data

Change the layout of the Tests table in the following ways:

- Use the buttons located to the right of the table to add, remove, or reorder tests.
- Drag column headers to reorder columns.

Enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Copying and Pasting Data

You can copy and paste data in the Tests table in the following ways:

- Press <Ctrl-C> to copy the contents of selected table rows, columns, or cells to the clipboard.
- Press <Ctrl-V> to insert data from the clipboard into the table starting from the top left selected cell. If the number of rows of data on the clipboard is greater than the number of tests in the Tests table, the paste command adds new tests to the step to match the clipboard data.

- You can copy data to tests on the same Semiconductor Multi Test or Semiconductor Sequence Call step, to tests on other Semiconductor Multi Test or Semiconductor Sequence Call steps, or to a Microsoft Excel spreadsheet.
- When you modify the data in Excel and paste it back to a Semiconductor Multi Test or Semiconductor Sequence Call step, the step returns an error if the modified data is invalid for a test.

Part Average Testing results

When used in a Part Average Testing step, the Tests tab displays, during test program execution, the results of the tests that it performs. If testing is not in progress, if the step has not yet executed, if a PAT plug-in is not installed, or if the Part Average Testing step type is disabled in the plug-in, the Tests tab does not display any content.

Columns

The Tests table contains the following columns:

- **Test Number**—The test number. If you do not specify a test number, the step assigns the value `0`. The step stores this value in the Step.Result.Evaluations[i].TestNumber property. For tests that correspond to a pin in a pin group, the step sets the test number to a value that equals the sum of the test number specified for the pin group test and the zero-based index of the pin in the pin group.

  TSM uses the test number and test name to identify the test in an STDF log file.
- **Test Name**—A string that identifies or describes the test. The step stores this string in the Step.Result.Evaluations[i].TestName property. For tests that correspond to a pin in a pin group, the step sets the test name to a string that consists of the test name specified for the pin group test concatenated with the pin name.
- **Pin**—The pin or pin group to test. The step stores this value in the Step.Result.Evaluations[i].Pin property.

  When you specify a valid pin map file in the Test Program Editor, this cell

contains a drop-down menu that includes the pin groups, DUT pins, and system pins the pin map defines. If the pin map file is invalid or if you do not specify a pin map file, this cell contains an editable string.

Leave the Pin column blank for tests that receive data from the Publish Pattern Results VI or `PublishPatternResults` .NET method.

If you specify a pin group, the Tests table inserts a test for each pin in the pin group. These inserted tests are not editable and are identical to the test that specifies a pin group, with the following exceptions:

- The Pin column is set to the name of a pin in the pin group.
- The Test Number column is computed by adding the test number specified for the pin group test to the zero-based index of the pin in the pin group.
- The Test Name column is computed by appending the name of the pin in the pin group to the test name specified for the pin group test.

Although the step does not save the inserted tests in the sequence file, at run time the step adds and performs tests identical to these inserted tests. The pin group test does not appear at run time or in log files or reports.

Note To use a different numbering scheme for test numbers or a different naming scheme for the test name for pins in a pin group, copy the tests that were inserted for the pin group, delete the pin group test, and paste the copied tests into the Tests table. You can then edit the test numbers and test names directly.

- **Published Data Id**—(Semiconductor Multi Test step) A string that identifies the measurement. The step stores this string in the Step.Result.Evaluations[i].MeasurementId property.

This option is not available when you specify an expression in the Test Data Source cell.

The published data ID string can be empty, but if the code module that the

step calls performs multiple measurements for a single pin, the code module must distinguish the different measurements using unique values for the published data ID when the code module uses the TSM Code Module API to publish the measurement data.

If a Semiconductor Multi Test step executes inside a sequence called by a Semiconductor Sequence Call step and the Semiconductor Sequence Call step defines a test with the same step name and published data ID, then the properties of the test in the Semiconductor Sequence Call step replace the properties of the test in the Semiconductor Multi Test.

- **Step Name.Published Data Id**—(Semiconductor Sequence Call step) A string that identifies the step name and published data ID of a Semiconductor Multi Test step in the sequence called by the Semiconductor Sequence Call step. The Semiconductor Sequence Call step stores this string in the Step.Result.Evaluations[i].MeasurementId property.

The Semiconductor Sequence Call step uses the step name and published data ID to assign the tests you specify to the Semiconductor Multi Test steps in the called sequence.

- **Low Limit**—A numeric expression that specifies the low limit. This option is available only when you select Numeric Limit in the Evaluation Type column. The step stores this expression in the Step.Result.Evaluations[i].NumericLimit.LowLimitExpr property.

If you specify a low limit, you must also specify a high limit. Use the **Evaluation Comparison Mode** option on the Options tab to configure how the step evaluates the limits.

If you do not specify a low limit and high limit, the step skips the comparison and logs only the measurement value in an STDF logfile.

- **High Limit**—A numeric expression that specifies the high limit. This option is available only when you select Numeric Limit in the Evaluation Type column. The step stores this expression in the Step.Result.Evaluations[i].NumericLimit.HighLimitExpr property.

If you specify a high limit, you must also specify a low limit. Use the **Evaluation Comparison Mode** option on the Options tab to configure how the step evaluates the limits.

If you do not specify a low limit and high limit, the step skips the comparison and logs only the measurement value in an STDF logfile.

- **Scaling Factor**—The scaling factor to use for the measurement and limit values. The step assumes that measurement values are in base units and that limit values are in scaled units. The Tests table displays all values in scaled units.
- **Base Units**—A string that specifies the base units associated with the limits and the measurement. This option is available only when you select Numeric Limit in the Evaluation Type column. The step stores this string in the Step.Result.Evaluations[i].NumericLimit.Units property.
- **Software Bin**—The test software bin. This bin must be a valid fail bin. The step stores this value in the Step.Result.Evaluations[i].FailBin property.

If the test fails and TSM has not yet assigned a software bin to the DUT, the step assigns this software bin to the DUT. If you have already assigned a bin using the Set and Lock Bin step and the test fails, the step reports a run-time error.

When you specify a valid bin definitions file in the Test Program Editor, this cell contains a drop-down menu that includes the software fail bins the bin definitions file defines. If the bin definitions file is invalid or if you do not specify a bin definitions file, this cell contains an editable string.

If you do not specify a software bin, the step stores the value `-1`, which specifies that no binning occurs for the test.

- **Evaluation Type**—The Numeric Limit, Pass/Fail, or None evaluation type. Use a Numeric Limit evaluation for a parametric test, and use a Pass/Fail evaluation for a functional test. Use the None evaluation type to publish data from a code module without evaluating the data. The step stores this value in the Step.Result.Evaluations[i].EvaluationType property.

When you select the Pass/Fail or None evaluation type, the Low Limit, High Limit, and Base Units cells are not available.

- **Test Data Source**—Optional. An expression that specifies the measurement data to use for the test. The step stores this expression in the Step.Result.Evaluations[i].MeasurementSourceExpr property.

  When you specify an expression, the Pin and Published Data Id cells are not available.

  If you do not specify an expression, you must use the TSM Code Module API in the code module the step calls to publish the measurement data to the test. This expression must be a numeric value for tests with a Numeric Limit evaluation type or must be a Boolean value for tests with a Pass/Fail evaluation type. This expression must be a Boolean, numeric, or string value for None evaluation type tests.

  This option is not available in the Test edit tab of the Semiconductor Sequence Call step.

- **Export Data To**—Optional. An expression that specifies a location to copy the measurement data from the test. Use this expression when you need to refer to the measurement in subsequent steps. The step stores this expression in the `Step.Result.Evaluations[i].MeasurementDestinationExpr` property. For tests that refer to a pin group, use an array expression to copy the measurement data for each pin test to an element in the array. The step automatically adjusts the size of the array to have enough elements to contain the data for each pin in the pin group.

The following columns are available only in the Execution window for a Semiconductor Multi Test or Semiconductor Sequence Call step suspended at a breakpoint during an execution:

- **Status**—The status of the test.
- **Data**—The measurement data associated with the test.

See Also

Exporting and Importing Test Limits with Text Files

Semiconductor Multi Test Step Constants

Semiconductor Multi Test Step Properties

Scaling Measurement and Limit Data (TSM)

Use the Scaling Factor column in the Tests table on the Tests tab of the Semiconductor Multi Test step and the Semiconductor Sequence Call step to specify the scaling factor for the base units for test limits and measurements.

Changing the scaling factor in the Scaling Factor column in the Tests table sets the same scaling factor for the limits and the measurement.

To specify different scaling factors for each limit and the measurement, use the Property Browser panel on the Properties tab of the Step Settings pane and set the following properties:

Property	Description
`Step.Result.Evaluations[i].NumericLimit.DataScalingExponent`	Measurement scaling factor
`Step.Result.Evaluations[i].NumericLimit.LowLimitScalingExponent`	Low limit scaling factor
`Step.Result.Evaluations[i].NumericLimit.HighLimitScalingExponent`	High limit scaling factor

The Semiconductor Multi Test step and the Semiconductor Sequence Call step use the scaling factors in the following ways:

- **Measurement Value Scaling Factor**—The steps assume the measurement values that code modules publish use base units. At run time, the steps use the measurement scaling factor to display the measurement value in scaled units in the Data column of the Tests table. The STDF Log result processing plug-in stores the corresponding scaling exponent in the RES_SCAL field of the PTRs. The Debug Test Results Log result processing plug-in uses the measurement scaling factor to format the values and units in the Measurement Value column of the Debug Test Results Log.

- **Limits Scaling Factors**—The steps assume the limit values in the Low Limit and High Limit columns use scaled units. At run time, the steps use the limit scaling factors to convert the limit values to base units to compare the measurement value to the limit values. The STDF Log result processing plug-in stores the corresponding scaling exponents for the low limit and high limit scaling factors in the LLM_SCAL and HLM_SCAL fields of the PTRs. The Debug Test Results Log result processing plug-in uses the limit scaling factors to add units to the values in the Low Limit and High Limit columns of the Debug Test Results Log.

Supported Scaling Factors

TSM recognizes the following scaling factors, which correspond to the recognized values for the RES_SCAL, HLM_SCAL, and LLM_SCAL fields of the STDF Parametric Test Records (PTRs):

Scaling Factor Name	Scaling Factor Value	Units Prefix	DataScalingExponent, LowLimitScalingExponent, and HighLimitScalingExponent Property Values	STDF_SCAL Field Value
femto	10	f	-15	15
pico	10	p	-12	12
nano	10	n	-9	9
micro	10	u	-6	6
milli	10	m	-3	3
percent	10	%	-2	2
<blank>	1	0	0	
kilo	10	K	3	-3
mega	10	M	6	-6

giga	10	G	9	-9
tera	10	T	12	-12

See Also

Exporting and Importing Test Limits with Text Files

Semiconductor Multi Test Part Average Testing Tab

When a part average testing (PAT) callbacks sequence file exists in the `<TestStand Public>\Components\Callbacks\NI_SemiconductorModule` directory, the Part Average Testing tab contains a table that lists PAT settings for the tests the step performs.

Entering Data

You can enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Copying and Pasting Data

You can copy and paste data in the table in the following ways:

- Press <Ctrl-C> to copy the contents of selected table rows, columns, or cells to the clipboard.
- Press <Ctrl-V> to insert data from the clipboard into the table starting from the top left selected cell. If the number of rows of data on the clipboard is

greater than the number of tests in the Tests table, the paste command adds new tests to the step to match the clipboard data.

- You can copy data to tests on the same Semiconductor Multi Test step, to tests on other Semiconductor Multi Test steps, or to a Microsoft Excel spreadsheet.
- When you modify the data in Excel and paste it back to a Semiconductor Multi Test step, the step returns an error if the modified data is invalid for a test.

Columns

The Part Average Testing table contains the following columns:

Notes

- The current PAT environment settings determine which of the following columns are visible.
- Part average testing is available only for tests with a Numeric Limit evaluation type.

- **Test Number**—The read-only test number for the test to which the PAT settings apply. This column is always visible.
- **Test Name**—The read-only test name of the test to which the PAT settings apply. This column is always visible.
- **PAT Base Test Number**—A base test number a PAT algorithm can use to automatically generate PAT test numbers. PAT plug-ins that use this setting typically hide and ignore the **Dynamic PAT Test Number** and **Static PAT Test Number** settings.
- **Enable Dynamic PAT**—Enables dynamic part average testing for the test and enables editing of the other dynamic PAT settings for the test. When this setting is enabled, TSM automatically generates a PAT test associated with the test. The PAT algorithm customizes, assigns limits to, and performs the dynamic PAT tests after all other tests in the `MainSequence` sequence complete or during the execution of a Perform Part Average Testing step.

- **Dynamic PAT Test Number**—The test number for the dynamic PAT test. This setting is available only when **Enable Dynamic PAT** is enabled.
- **Dynamic PAT Test Name**—The test name for the dynamic PAT test. This setting is available only when **Enable Dynamic PAT** is enabled.
- **Dynamic PAT Software Bin**—The software bin for the dynamic PAT test. This bin must be a valid fail bin defined in the bin definitions file. This setting is available only when **Enable Dynamic PAT** is enabled.
- **Dynamic PAT Low Limit**—The low limit for the dynamic PAT test specified in standard deviations from the mean. A typical PAT algorithm uses this setting to convert the test measurement data to standard deviations before performing the PAT test. This setting is available only when **Enable Dynamic PAT** is enabled.
- **Dynamic PAT High Limit**—The high limit for the dynamic PAT test specified in standard deviations from the mean. A typical PAT algorithm uses this setting to convert the test measurement data to standard deviations before performing the PAT test. This setting is available only when **Enable Dynamic PAT** is enabled.
- **Enable Static PAT**—Enables static part average testing for the test and enables editing of the other static PAT settings for the test. When this setting is enabled, TSM automatically generates a static PAT test associated with the test. The PAT algorithm obtains static PAT limits, sets the limits on the static PAT tests, and performs the static PAT tests after all other tests in the `MainSequence` sequence complete or when a Perform Part Average Testing step executes.
- **Static PAT Test Number**—The test number for the static PAT test. This setting is available only when **Enable Static PAT** is enabled.
- **Static PAT Test Name**—The test name for the static PAT test. This setting is available only when **Enable Static PAT** is enabled.
- **Static PAT Software Bin**—The software bin for the static PAT test. This bin must be a valid fail bin defined in the bin definitions file. This setting is available only when **Enable Static PAT** is enabled.

See Also

Part Average Testing

Semiconductor Multi Test Per-Site Inputs Tab

The Per-Site Inputs tab contains a table that lists the site-specific inputs to the code module for the step. Use per-site inputs when the data you need to send to the code module has the potential to be different for each site. Specify the data value in the Per-Site Inputs table and use the TSM Code Module API in the code module to obtain the data.

Modifying the Layout and Entering Data

You can modify the layout of the Per-Site Inputs table in the following ways:

- Use the buttons located to the right of the table to add, remove, or reorder per-site inputs.
- Drag column headers to reorder columns.

You can enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Columns

The Per-Site Inputs table contains the following columns:

- **Pin**—The pin associated with the data. If you do not specify a pin, you must specify a non-empty **Input Data Id** string.

  When you specify a valid pin map file on the Pin Map panel of the Test

Program Editor, this cell contains a drop-down menu that includes the DUT and system pins the pin map defines. If the pin map file is invalid or if you do not specify a pin map file, this cell contains an editable string.

- **Input Data Id**—A string that identifies the input data. If you do not specify a value for this string, you must specify a pin.
- **Data Source**—An expression that specifies the input data to send to the code module. The expression must evaluate to a Boolean, number, or string value.

The order of the per-site inputs in the table is not significant.

Semiconductor Multi Test Options Tab

The Options tab contains the following options:

- **Stop Performing Tests after First Failure**—When you enable this checkbox and a test fails, the Semiconductor Multi Test does not evaluate any subsequent tests in the step and sets the remaining test results to `Skip`.
- **Evaluation Comparison Mode**—Select how the step compares limits for all tests the step specifies. This drop-down menu is available only when you specify at least one test with a low and high limit on the Tests tab.
- **Test Numeric Display Format**—Displays the TestStand numeric format associated with the limits and measurement data. The numeric format determines how the Tests tab displays numeric values and how the TestStand report generators display numeric values in reports. Click the **Edit** button to launch the Numeric Format dialog box, in which you can specify the numeric format.
- **Multisite Option**—Use this drop-down menu to specify how the step executes the step code module on multiple sites:
  - **One thread per subsystem**—Execute the code module for each subsystem in a separate thread. The Semiconductor Multi Test step identifies subsystems by using the pin map and the pins and relays shown in **SemiconductorModuleContext Pins and Relays**.

- **One thread only**—Execute the code module for all sites in a single thread.
- **One thread per site**—Execute the code module for each site in a separate thread. Use this option only when the code module does not use hardware shared among multiple sites.

Notes

- TSM does not synchronize executions when you use the Parallel process model
- Semiconductor Multi Test steps in the ProcessSetup and ProcessCleanup sequences always execute as if you specified **One thread only** in the **Multisite Option** drop-down menu.

- **Multisite Execution Diagram**—When you associate a pin map file with the sequence file, this diagram shows the threads the step uses to execute code modules when executing with the Batch process model using multiple sites. Each rectangle represents a single thread. The numbers within the rectangle represent the sites tested in the same thread. Each thread corresponds to one of the test socket threads. The step determines which test socket thread executes the code module at run time. This diagram is available only when you chose **Select Manually** from the **Specify Pins and Relays** drop-down menu.
- **Specify Pins and Relays**—Specifies which pins and relays are included in the `SemiconductorModuleContext`. Specify pins and relays manually or by using an expression.
- **SemiconductorModuleContext Pins and Relays**—Shows the list of pins and relays or specifies the expression that determines the pins and relays that the `SemiconductorModuleContext` object contains at run time.

By default, the `SemiconductorModuleContext` object contains all DUT pins in the pin map file and no system pins. If your code module uses any system pins, you must manually select the **Include System Pins** checkbox or include the system pins in your expression. Code modules that use system

pins execute in one thread for all sites. If your code module uses only DUT pins, you can improve performance by specifying only the DUT pins you use in the code module.

By default, the `SemiconductorModuleContext` object contains no relays. If your code module uses any system relays, you must manually select the **Include System Relays** checkbox or include the system relays in your expression. If your code module uses site relays, you must manually select the **Specify Site Relays** checkbox and specify the relays or include the site relays in your expression.

To specify the pins and relays manually, choose **Select Manually** from the **Specify Pins and Relays** drop-down menu and enable the options you want.

- **Include System Pins**—Include all system pins.
- **Specify DUT Pins**—Select the DUT pins or pin groups you want to include. When you select a pin group, the tab dims the DUT pins that belong to the pin group.
- **Include System Relays**—Include all system relays.
- **Specify Site Relays**—Select the relays, relay groups, or relay configurations you want to include. When you select a relay group or relay configuration, the tab dims the relays that belong to the relay group or configuration.

Note If you choose **One thread only** or **One thread per site** in the **Multisite Option** drop-down menu, all pins and relays are included in the `SemiconductorModuleContext` and the checkboxes are disabled.

To specify the pins and relays using an expression, select **Use Expression** from the **Specify Pins and Relays** drop-down menu. At run time, the expression you use must evaluate to an array of DUT and system pin names, and site and system relay names.

Note The `SemiconductorModuleContext` object for steps in the ProcessSetup and ProcessCleanup sequences always contains all DUT and system pins and all site and system relays.

See Also

Multisite Programming Techniques

Semiconductor Action Step

Use the Semiconductor Action step to perform an action, such as instrument configuration, with access to the pin map and per-site inputs. You can configure multisite and per-site input options directly on the step.

Note You cannot use multiple Semiconductor Action steps configured to use multiple threads in While loops, in Do While loops or in For loops that use the Custom Loop option when performing multisite testing. The Semiconductor Action step reports a run-time error in these situations. Use other types of loops instead, such as For loops that use the Fixed Number of Iterations option.

Configuring the Step

Use the Semiconductor Action step edit tabs in the TestStand Sequence Editor to specify the multisite and per-site input options.

Step Properties

The Semiconductor Action step type defines a set of step properties and constants, in addition to the built-in properties common to all TestStand steps.

Semiconductor Action Step Execution Overview

The Semiconductor Action step uses the following process when executing:

1. Creates a `SemiconductorModuleContext` object and stores it on the Step.SemiconductorModuleContext property if the site that executes the step is the working site. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system.
2. Calls a code module that completes the following actions:
    a. Receives the `Step.SemiconductorModuleContext` property as an input parameter.
    b. Obtains instrument channel and session information using the TSM Code Module API.
    c. Performs an action with access to the `SemiconductorModuleContext` and any per-site inputs specified for the step.

Note The Publish Data VI and Publish .NET method have no effect in code modules you call from the Semiconductor Action step because no test locations exist for publishing data.

Semiconductor Action Step Properties

The Semiconductor Action step type defines the following step properties:

- **Result.SemiconductorCommon**—A container that stores run-time data about the step. The NI_SemiconductorModule_SemiconductorCommon data type defines the following field:
    - **Sites**—A string that stores a comma-separated list of sites tested in the current thread, as shown in the Multisite Execution Diagram of the step. The step sets this property at run time.
    - **RaisedAlarms**—A string that stores a comma-separated list of alarms that were raised by the step. The step sets this property at run time.
- **SemiconductorModuleContext**—An object reference that stores the `SemiconductorModuleContext` object the step code module uses. The `SemiconductorModuleContext` object describes a subset of pins, relays, sites, and instruments on a test system. Code modules use this property to access the TSM Code Module API.

- **Multisite**—A container that stores the multisite options you configure for the step on the Options edit tab. The NI_SemiconductorModule_Multisite data type defines the following fields:
  - **SynchronizationOption**—The multisite option. The default option is one thread per subsystem.
  - **SpecifyPinsAndRelaysOption***—The option for how you specify what pins and relays to include in the `SemiconductorModuleContext` object for the step.
  - **PinsAndRelaysExpression***— The expression that specifies what pins and relays to include in the `SemiconductorModuleContext` object for the step. The step uses this expression only when the `SpecifyPinsAndRelaysOption` property is set to 1.
  - **DUTPinFilterOption***—The option for determining what pins to include in the `SemiconductorModuleContext` object for the step. By default, all DUT pins are included. If you try to access a pin that is not included in the `SemiconductorModuleContext` object, the step returns an error.
  - **IncludedDUTPins***—An array that stores the DUT pins to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `DUTPinFilterOption` property is set to 1.
  - **ExcludedDUTPins***—An array that stores the DUT pins to exclude from the `SemiconductorModuleContext` object for the step. The step uses this array only when the `DUTPinFilterOption` property is set to 0.
  - **IncludeSystemPins***—A Boolean value that specifies whether to include all system pins in the `SemiconductorModuleContext` object for the step.
  - **IncludedPinGroups***—An array that stores the pin groups to include in the `SemiconductorModuleContext` object for the step.
  - **SpecifySiteRelays***—The option to manually specify which site relays to include in the `SemiconductorModuleContext` object for the step. By default, no site relays are included.

- **IncludedSiteRelays***—An array that stores the site relays to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.
- **IncludedRelayGroups***—An array that stores the relay groups to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.
- **IncludedRelayConfigurations***—An array that stores the relay configurations to include in the `SemiconductorModuleContext` object for the step. The step uses this array only when the `SpecifySiteRelays` property is set to True.
- **IncludeSystemRelays***—A Boolean value that specifies whether to include all system relays in the `SemiconductorModuleContext` object for the step.

- **DotNetRuntimeData**—A hidden object reference that substeps of the Semiconductor Action step use to cache run-time information about the step. Do not directly interact with this property.

Note * The Semiconductor Action step uses these properties only when the **SynchronizationOption** is set to **One thread per subsystem** and the step is executing in a test socket thread. Steps in process model callback sequences such as `ProcessSetup` and `ProcessCleanup` include all DUT pins and system pins, and all site and system relays.

Semiconductor Action Step Constants

Use the following constants with the Semiconductor Action step properties to configure options for the step:

- **SynchronizationOption**—Use the following constants with the Step.Multisite.SynchronizationOption property:

- `0`—One thread per site
- `1`—One thread only
- `2`—One thread per subsystem

- **SpecifyPinsAndRelaysOption**—Use the following constants with the Step.Multisite.SpecifyPinsAndRelaysOption property:
  - `0`—Specify manually
  - `1`—Use expression
- **DUTPinFilterOption**—Use the following constants with the Step.Multisite.DUTPinFilterOption property:
  - `0`—Include all DUT pins except those listed in the ExcludedDUTPins array.
  - `1`—Exclude all DUT pins except those listed in the IncludedDUTPins array.
- **SpecifySiteRelays**—Use the following constants with the Step.Multisite.SpecifySiteRelays property:
  - `False`—Include all site relays.
  - `True`—Include only the site relays listed in the IncludedSiteRelays array.

Semiconductor Action Edit Tabs

Use the Semiconductor Action step edit tabs in the TestStand Sequence Editor to specify the multisite and per-site input options for the Semiconductor Action step.

The Step Settings pane for the Semiconductor Action step contains the following tabs:

- Per-Site Inputs—Configure the per-site inputs for the step. Each per-site input specifies a data value that you access in a code module using the TSM Code Module API.
- Options—Configure multisite and other options for the step.

Semiconductor Action Per-Site Inputs Tab

The Per-Site Inputs tab contains a table that lists the site-specific inputs to the code module for the step. Use per-site inputs when the data you need to send to the code module has the potential to be different for each site. Specify the data value in the Per-Site Inputs table and use the TSM Code Module API in the code module to obtain the data.

Modifying the Layout and Entering Data

You can modify the layout of the Per-Site Inputs table in the following ways:

- Use the buttons located to the right of the table to add, remove, or reorder per-site inputs.
- Drag column headers to reorder columns.

You can enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Columns

The Per-Site Inputs table contains the following columns:

- **Pin**—The pin associated with the data. If you do not specify a pin, you must specify a non-empty **Input Data Id** string.

  When you specify a valid pin map file on the Pin Map panel of the Test Program Editor, this cell contains a drop-down menu that includes the DUT and system pins the pin map defines. If the pin map file is invalid or if you do not specify a pin map file, this cell contains an editable string.

- **Input Data Id**—A string that identifies the input data. If you do not specify a value for this string, you must specify a pin.
- **Data Source**—An expression that specifies the input data to send to the code module. The expression must evaluate to a Boolean, number, or string value.

The order of the per-site inputs in the table is not significant.

Semiconductor Action Options Tab

The Options tab contains the following options:

- **Multisite Option**—Select one of the following options to specify how the step executes the step code module on multiple sites:
  - **One thread per subsystem**—Execute the code module for each subsystem in a separate thread. The Semiconductor Action step identifies subsystems by using the pin map and the pins shown in **SemiconductorModuleContext Pins and Relays**.
  - **One thread only**—Execute the code module for all sites in a single thread.
  - **One thread per site**—Execute the code module for each site in a separate thread. Use this option only when the code module does not use hardware shared among multiple sites.

Notes

- TSM does not synchronize executions when you use the Parallel process model
- Semiconductor Action steps in the ProcessSetup and ProcessCleanup sequences always execute as if you specified **One thread only** in the **Multisite Option** drop-down menu.

- **Multisite Execution Diagram**—When you associate a pin map file with the sequence file, this diagram shows the threads the step uses to execute

code modules when executing with the Batch process model using multiple sites. Each rectangle represents a single thread. The numbers within the rectangle represent the sites tested in the same thread. Each thread corresponds to one of the test socket threads. The step determines which test socket thread executes the code module at run time. This diagram is available only when you chose **Select Manually** from the **Specify Pins and Relays** drop-down menu.

- **Specify Pins and Relays**—Specifies which pins and relays are included in the `SemiconductorModuleContext`. Specify pins and relays manually or by using an expression.
- **SemiconductorModuleContext Pins and Relays**—Shows the list of pins and relays or specifies the expression that determines the pins and relays that the `SemiconductorModuleContext` object contains at run time.

By default, the `SemiconductorModuleContext` object contains all DUT pins in the pin map file and no system pins. If your code module uses any system pins, you must manually select the **Include System Pins** checkbox or include the system pins in your expression. Code modules that use system pins execute in one thread for all sites. If your code module uses only DUT pins, you can improve performance by specifying only the DUT pins you use in the code module.

By default, the `SemiconductorModuleContext` object contains no relays. If your code module uses any system relays, you must manually select the **Include System Relays** checkbox or include the system relays in your expression. If your code module uses site relays, you must manually select the **Specify Site Relays** checkbox and specify the relays or include the site relays in your expression.

To specify the pins and relays manually, choose **Select Manually** from the **Specify Pins and Relays** drop-down menu and enable the options you want.

- **Include System Pins**—Include all system pins.

- **Specify DUT Pins**—Select the DUT pins or pin groups you want to include. When you select a pin group, the tab dims the DUT pins that belong to the pin group.
- **Include System Relays**—Include all system relays.
- **Specify Site Relays**—Select the relays, relay groups, or relay configurations you want to include. When you select a relay group or relay configuration, the tab dims the relays that belong to the relay group or configuration.

Note If you choose **One thread only** or **One thread per site** in the **Multisite Option** drop-down menu, all pins and relays are included in the `SemiconductorModuleContext` and the checkboxes are disabled.

To specify the pins and relays using an expression, select **Use Expression** from the **Specify Pins and Relays** drop-down menu. At run time, the expression you use must evaluate to an array of DUT and system pin names, and site and system relay names.

Note The `SemiconductorModuleContext` object for steps in the ProcessSetup and ProcessCleanup sequences always contains all DUT and system pins and all site and system relays.

See Also

Multisite Programming Techniques

Semiconductor Sequence Call Step

Use the Semiconductor Sequence Call step to call a sequence and assign tests to Semiconductor Multi Test steps in the called sequence. Call a sequence in the current sequence file or in another sequence file. The Semiconductor Sequence Call step uses the Sequence Adapter.

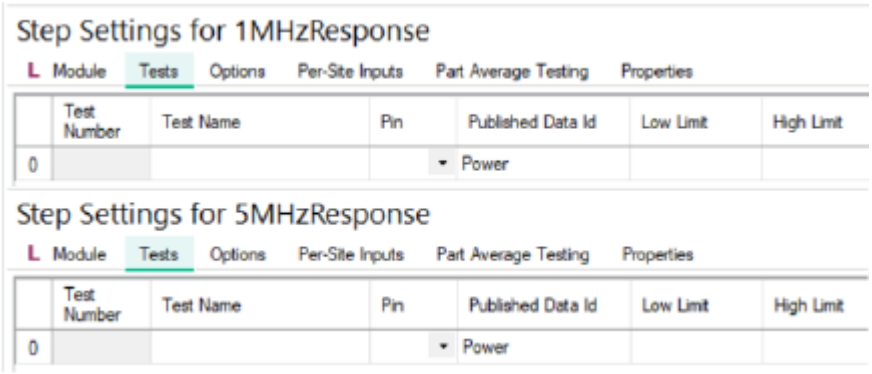
Note The Semiconductor Sequence Call step does not evaluate tests. It defines and assigns tests to the Semiconductor Multi Test steps in the called sequence. The Semiconductor Multi Test steps in the called sequence evaluate the tests.

Configuring the Step

Complete the following steps to create a subsequence with the Semiconductor Multi Test steps you want to call from a Semiconductor Sequence Call step.

1. Create a new sequence.
2. Insert one or more Semiconductor Multi Test steps in the sequence. The figure below shows a subsequence named `MeasureGain` that consists of one Semiconductor Action step and two Semiconductor Multi Test steps.

3. On the Module tab of the Step Settings pane, specify the code modules you want to use for each step.
4. On the Tests tab, add tests and specify the published data ID for each test.



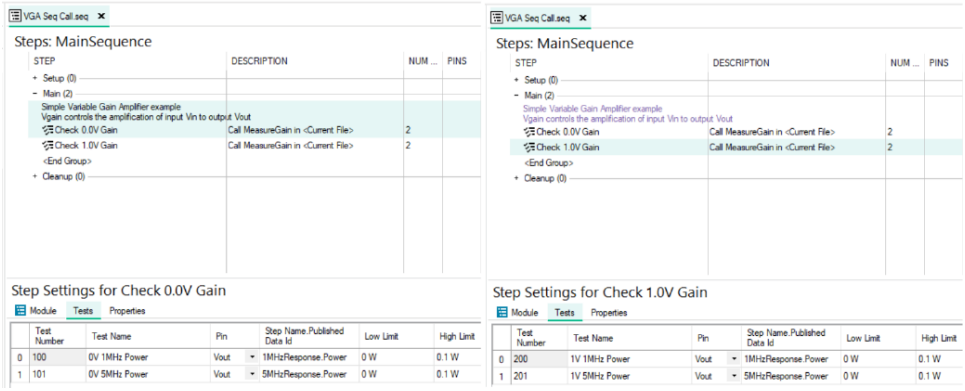
5. Add and configure any other required steps, such as Semiconductor Action steps.
6. Save the sequence file.

Complete the following steps to configure a Semiconductor Sequence Call step.

1. Insert one or more Semiconductor Sequence Call steps into a sequence where you want to call the subsequence you created.
2. For each Semiconductor Sequence Call step, select the Module tab of the Step Settings pane and specify the sequence file that contains the subsequence.
3. Configure any additional Sequence Call Module options in the Module tab.
4. Select the Tests tab of the Step Settings pane and configure the parameters you want to assign to each Semiconductor Multi Test step called by the Semiconductor Sequence Call step.

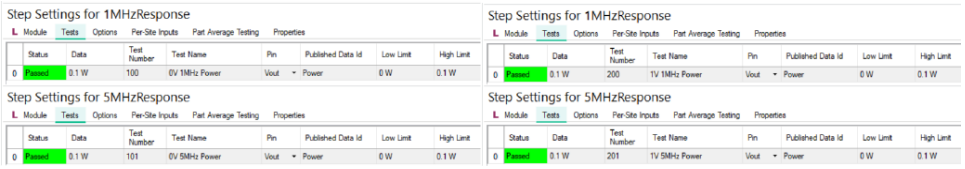
    a. Define the Step Name.Published Data Id.
    The value you specify must correspond to the step name and published data ID of the Semiconductor Multi Test step for which you want to specify parameters. The Semiconductor Sequence Call step assigns parameters to the Semiconductor Multi Test step with a step name and publish data ID that match the Step Name.Published Data Id you define.

    The figure below shows the `MainSequence` sequence which consists of two Semiconductor Sequence Call steps named `Check 0.0V Gain` and `Check 1.0V Gain`. Each Semiconductor Sequence Call step consists of two tests that specify `1MHzResponse.Power` and `5MHzResponse.Power` in the Step Name.Published Data Id field. These values correspond to the names and published data IDs of the Semiconductor Multi Test steps shown in the figures in Step 2 and Step 4 above.



b. Define test number, test name, pin or pin group, limits, scaling factor, base units, software bin, evaluation type, and measurement destination.

5. Run the TestStand Sequence Analyzer and resolve any errors or warnings. Semiconductor Sequence Call steps with unresolved Sequence Analyzer errors will throw run-time errors.
6. Run the sequence. Before TSM executes each Semiconductor Multi Test step, the Semiconductor Sequence Call step assigns the test parameters you specified to the Semiconductor Multi Test steps.
   The figure below shows the Semiconductor Multi Test steps called by Semiconductor Sequence Call steps `Check 0.0V Gain` and `Check 1.0V Gain` on the left and right, respectively. The parameters for each step have been assigned by the corresponding Semiconductor Sequence Call step.



Step Properties

The Semiconductor Sequence Call step defines a set of step properties, in addition to the built-in properties common to all TestStand steps.

See Also

Test Program Editor

Exporting and Importing Test Limits with Text Files

Semiconductor Sequence Call Step Properties

The Semiconductor Sequence Call step defines the following properties. These properties are assigned to the Semiconductor Multi Test step tests in the called sequence that have corresponding step names and published data IDs.

TSM replaces all step property values for the Semiconductor Multi Test steps in the called sequence, except `MeasurementSourceExpr` and `MeasurementDestinationExpr`, with the values specified in the Semiconductor Sequence Call step.

- **Result.Evaluations**—An array that stores the tests you configure for the step on the Tests edit tab. The NI_SemiconductorModule_Evaluation data type defines the following fields:
  - **EvaluationType**—The type of evaluation the test performs. The default evaluation type is Numeric Limit.
  - **Pin**—A string that stores the test pin or pin group.
  - **MeasurementId**—A string that stores the Step Name.Published Data Id value. The Semiconductor Sequence Call step uses the step name and published data ID to assign the tests you specify to the Semiconductor Multi Test steps in the called sequence.
  - **TestNumberExpr**—An expression that determines the test number at run time. If you specify this expression, the Semiconductor Multi Test step you call from the Semiconductor Sequence Call step evaluates the expression and copies the evaluated value to its `TestNumber` property.
  - **TestNumber**—A number that stores the test number.
  - **FailBinExpr**—An expression that determines the software fail bin for the test at run time. The expression must evaluate to a valid software bin number. If you specify this expression, the Semiconductor Multi Test step you call from the Semiconductor Sequence Call step evaluates the expression and copies the evaluated value to the `FailBin` property.
  - **FailBin**—A number that stores the software fail bin for the test.

- **TestNameExpr**—An expression that determines the test name at run time. If you specify this expression, the Semiconductor Multi Test step you call from the Semiconductor Sequence Call step evaluates the expression and copies the evaluated value to its `TestName` property.
- **TestName**—A string that stores the test name.
- **MeasurementSourceExpr**—The value of this property is not used by the Semiconductor Sequence Call step and it is not passed to Semiconductor Multi Test steps. The value cannot be set on the Tests edit tab of the Semiconductor Sequence Call step.
- **MeasurementDestinationExpr**—Optional. An expression that specifies a custom location in the context of the Semiconductor Sequence Call step to store the numeric or Boolean measurement value. The Semiconductor Multi Test step stores the measurement value in the locations specified by the MeasurementDestinationExpr properties of the Semiconductor Multi Test step and the Semiconductor Sequence Call step after the step performs the test.
- **Status**—This property is not used by the Semiconductor Sequence Call step.
- **NumericLimit**—The NI_SemiconductorModule_Evaluation_NumericLimit data type defines the following fields:
  - **Data**—This property is not used by the Semiconductor Sequence Call step.
  - **LowLimitExpr**, **LowLimit**, **HighLimitExpr**, and **HighLimit**—The limits for the test.
  - **ComparisonType**—The type of comparison, such as `"GELE"`.
  - **Units**—A string that stores the base units for the limits and measurement value.
  - **DataScalingExponent, LowLimitScalingExponent, HighLimitScalingExponent**—Numbers that determine the scaling factors for the measurement value and each limit value. Each of these

properties contains an integer value that TSM uses as an exponent for the number 10 to determine the scaling factor.

- **CorrelationOffset**—The correlation offset value to apply to test results on a per-site basis at run time before evaluating the test result data against limits. The Load Correlation Offsets step sets this property.
- **SimulatedData**—The SimulatedData container uses the following fields to specify values for tests in Offline Mode:
  - **SimulatedDataUsageType**—Set to **AllSites** to specify a single value for all sites. Set to **PerSite** to specify specific values for each site. Set to **UsePublishedValue** if you want to use the published values. The default value is **None**, which specifies the default behavior of Offline Mode to ignore the values of **AllSite**, **PerSite**, and the published values.
    - **AllSites**—Specifies a single value for all sites.
    - **PerSite**—Specifies specific values for each site.

- **PassFail**—The NI_SemiconductorModule_Evaluation_PassFail data type defines the following fields:
  - **Data**—This property is not used by the Semiconductor Sequence Call step.
  - **SimulatedData**—The SimulatedData container uses the following fields to specify values for tests in Offline Mode:
    - **SimulatedDataUsageType**—Set to **AllSites** to specify a single value for all sites. Set to **PerSite** to specify specific values for each site. Set to **UsePublishedValue** if you want to use the published values. The default value is **None**, which specifies the default behavior of Offline Mode to ignore the values of **AllSite**, **PerSite**, and the published values.
      - **AllSites**—Specifies a single value for all sites.
      - **PerSite**—Specifies specific values for each site.

- **DotNetRuntimeData**—A hidden object reference that sub-steps of the Semiconductor Sequence Call step use to cache run-time information about the step. Do not directly interact with this property.

Notes

- The numeric representation for the numeric limit data and limit values must be double-precision, 64-bit floating-point values, and cannot be signed or unsigned 64-bit integers.
- The `TestNameExpr`, `TestNumberExpr`, and `FailBinExpr` properties do not correspond to options on the Semiconductor Multi Test step edit tabs.

Set Relays Step (TSM)

Use the Set Relays step to control relays and to apply relay configurations.

Configuring the Step

Use the Set Relays Step edit tab in the TestStand Sequence Editor to specify how you want to control relays and apply relay configurations.

Set Relays Tab (TSM)

The Set Relays tab contains the following options:

- **Relay or Relay Configuration Name**—Specifies the relay or relay group that you want to control, or the relay configuration that you want to apply. Relays, relay groups, and relay configurations must be defined in the pin map file.
  Select the **Edit Pin Map** button next to the input field to launch the Pin Map Editor and edit relays, relay groups, and relay configurations in the pin map file.

Initialize NI-SWITCH relay driver sessions for all relay driver modules in the pin map file and register these sessions with the Set Relay Driver NI-SWITCH Session VI or the `SetRelayDriverNISwitchSession` .NET method.

- **Relay Action**—Specifies how you want to set the relays. Valid options include:
  - **Apply Relay Configuration**—Applies the relay configuration specified in **Relay or Relay Configuration Name**. Relays are opened or closed according to the relay position specified in the relay configuration.
    This option is only valid if **Relay or Relay Configuration Name** specifies a relay configuration.
  - **Open Relays**—Opens the relays specified in **Relay or Relay Configuration Name**. If **Relay or Relay Configuration Name** specifies a relay group, all relays in the group are opened.
    This option is only valid if **Relay or Relay Configuration Name** specifies a relay or relay group.
  - **Close Relays**—Closes the relays specified in **Relay or Relay Configuration Name**. If **Relay or Relay Configuration Name** specifies a relay group, all relays in the group are closed.
    This option is only valid if **Relay or Relay Configuration Name** specifies a relay or relay group.
- **Time to wait (in seconds)**—Specifies the time to wait for the relays to settle after switching.

See Also

Set Relays Step Properties

Set Relays Step Properties

The Set Relays step type defines the following step properties:

- **RelayName**—The name of the relay or relay configuration to set.
- **RelayNameExpr**—An expression that determines at run time the name of the relay or relay configuration to set. If this expression is not empty, the Set Relays step evaluates the expression and substitutes the evaluated value for the `RelayName` property.
- **CloseRelay**—A flag that specifies the action for a relay or relay group. When False, individual relays or relays in a relay group are opened. When True, individual relays or relays in a relay group are closed.
This flag applies only when `RelayName` or `RelayNameExpr` specify a relay or relay group.
- **CloseRelayExpr**—An expression that determines at run time the action for a relay or relay group. If this expression is not empty, the Set Relays step evaluates the expression and substitutes the evaluated value for the `CloseRelay` property. When the expression evaluates to False, individual relays or relays in a relay group are opened. When the expression evaluates to True, individual relays or relays in a relay group are closed.
This expression applies only when `RelayName` or `RelayNameExpr` specify a relay or relay group.
- **TimeToSettle**—An expression that determines at run time the time in seconds that the Set Relay step waits for relays to settle after switching.

Get Test Information Step (TSM)

Use the Get Test Information step to obtain the values for lot settings, station settings, STS state, execution data, and custom test conditions. Store the value of the items in TestStand local variables so that any step in the sequence can access the values.

Configuring the Step

Use the Get Test Information edit tab in the TestStand Sequence Editor to specify the list of lot settings, station settings, STS state, execution data, and custom test conditions and the locations to store the values.

Get Test Information Tab (TSM)

The Get Test Information tab contains a table that lists the items the step accesses.

Modifying the Layout and Entering Data

Use the buttons located to the right of the table to add, remove, or reorder per-site inputs.

You can enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Columns

The test information table contains the following columns:

- **Filter**—Drop-down menu of the category types for the item you want to access. Select **All** to display items from all the categories. Categories include the following:

  Use this **Toggle Expansion** button to expand or collapse all the following sections at once. Use the black arrows next to each heading to expand or collapse sections individually. You must expand each section you want to print or search.

  ▾ Lot and Station Settings

  You can access the standard lot and station settings and custom lot and station settings you create from this step.

  ▾ STS Test Head

The following table lists the STS properties you can access from this step. Except for the `TestHeadAvailable` property, these queries must be run only on an STS with STS Maintenance Software 19.0 or later.

Name	Type	Description
12VPowerSupplyEnabled	Boolean	Returns whether the 12 V pins on STS CX and STS DX systems are enabled. The pins are disabled by default at STS power up and when a load board is not installed or locked down on the STS, such as when the value of the `DIBPresent` property is `False`. These signals are available on the system cable for STS CX systems and on the P143 system block for STS DX systems.  **Note** You can successfully make this query only on STS M2 systems. Otherwise, the step returns a run-time error.
48VPowerSupplyEnabled	Boolean	Returns whether the 48 V pins on STS CX and STS DX systems are enabled. The pins are disabled by default at STS power up and when a load board is not installed or locked down on the STS, such as when the value of the `DIBPresent` property is `False`. These signals are available on the system cable for STS CX systems and on the P143 system block for STS DX systems.

		Note The step returns a run-time error if you make this query on STST1 M1 systems.
CurrentRunTime	String	Reads from the system controller of the STS the amount of time the STS has been running since the last power-on event.
DIBLocked	Boolean	Returns the locked state of the DIB locker.
DIBPresent	Boolean	Returns whether a load board is installed by detecting whether the load board shorts the DIB_INTERLOCK pin to GND. The DIB_INTERLOCK and GND signals are available on the system cable for STS CX systems and on the P143 system block for STS DX systems. **Note** You must design the load board to connect DIB_INTERLOCK to GND to activate STS DIB detection. Refer to the **DIB Design Kit** for more information about designing load boards.
FramePartNumber	String	Returns the unique identifier and revision of the STS.

FrameSerialNumber	String	Returns the unique serial number of the STS.
LoadBoardIDPROM.ChecksumPassed	Boolean	Returns whether the checksum of the load board passed.
LoadBoardIDPROM.DateManufactured	String	Returns the manufacture date of the load board.
LoadBoardIDPROM.Description	String	Returns the description of the load board.
LoadBoardIDPROM.Format	Number	Returns the version number of the format that describes how the raw data from the IDPROM is converted into IDPROM values.
LoadBoardIDPROM.Manufacturer	String	Returns the manufacturer of the load board.
LoadBoardIDPROM.PartNumber	String	Returns the part number of the load board. By default, TSM writes this property to the `LoadBoardType` station settings and logs the value to the corresponding STDF record field if `LoadBoardIDPROM.ChecksumPassed` is `true`. You can override this behavior in the `GetStationSettings` and `ConfigureStationSettings` callback sequences of the `SemiconductorModuleCallbacks.seq` sequence file.
LoadBoardIDPROM.Revision	Number	Returns the revision number of the load board.
LoadBoardIDPROM.SerialNumber	Number	Returns the serial number of the load board. By default, TSM writes this property to the `LoadBoardId` station settings and logs the value to the corresponding STDF record field if `LoadBoardIDPROM.C`

		hecksumPassed is true. You can override this behavior in the GetStationSettings and ConfigureStationSettings callback sequences of the SemiconductorModuleCallbacks.seq sequence file.
SystemPartNumber	String	Returns the unique identifier and revision of the STS configuration.
SystemSerialNumber	String	Returns the unique serial number of the STS configuration. **Note** If the STS does not return a serial number or the serial number is **NOTSET**, this property returns the same value as the FrameSerialNumber property.
TestHeadAvailable	Boolean	Returns whether TSM can access the STS. The following conditions prevent TSM from accessing the STS: ▪ STS Maintenance Software is not installed. ▪ The version of STS Maintenance Software is earlier than version 19.0. ▪ An error occurred when trying to communicate with the STS.

TotalRunTime	String	Reads from the system controller of the STS the amount of time the STS has run over its entire lifetime.

Execution Data

The following table lists the execution data properties you can access from this step.

Name	Type	Description	Access Restrictions
BinType	Number or NI_SemiconductorModule_BinType	Type of the bin assigned to the part tested on the current site. (0=Pass, 1=Fail, 2=Other)	Test Socket Threads Only Valid in PostUUT
BinTypes	Array of numbers or array of NI_SemiconductorModule_BinType	The types of the bins assigned to each part on each site. (0=Pass, 1=Fail, 2=Other)	Valid in PostBatch
CSVTestResultsLogFilePath	String	Returns the absolute file path of the last CSV Test Results Log generated for the current lot.	Valid in MainSequence
CSVTestResultsLogFilePaths	Array of strings	Returns the absolute file paths of all CSV Test Results Logs generated for the current lot. If the Generate One File per Wafer option is enabled, one path is returned for each wafer processed in the lot so far. Otherwise, the length of the array is 1.	Valid in MainSequence
DebugTestResultsLogFilePath	String	Returns the absolute file path of the Debug Test Results Log gene	Valid in MainSequence

		rated for the current site.	
DebugTestResultsLogFilePaths	Array of strings	Returns the absolute file paths of the Debug Test Results Logs generated for all sites in the current lot.	Controller Thread Only
DidInlineQABlockExecute	Boolean	Returns true if an InlineQA step executed its block of steps for this site.	Test Socket Threads Only Valid in MainSequence
DieCoordinateX	Number	Returns the X wafer coordinate of the current die being tested on this site. The handler/prober driver sets this property value by setting values in the WaferRuntimeData.SiteDieCoordinates parameter in the StartOfTest handler driver entry point sequence. This property corresponds to the X_COORD field in the Part Results Record (PRR) in the STDF log file.	Test Socket Threads Only Valid in MainSequence
DieCoordinateY	Number	Returns the Y wafer coordinate of the current die being tested on this site. The handler/prober driver sets this property value by setting values in the WaferRuntimeData.SiteDieCoordinates parameter in the StartOfTest handler driver entry point sequence. This pr	Test Socket Threads Only Valid in MainSequence

		operty corresponds to the Y_COORD field in the Part Results Record (PRR) in the STDF log file.	
HardwareBinName	String	Returns the name of the hardware bin that the tester assigned to the last part that was tested on this site.	Test Socket Threads Only Valid in PostUUT
HardwareBinNames	Array of Strings	The names of the hardware bins that the tester assigned to the parts in the current batch.	Valid in PostBatch
HardwareBinNumber	Number	The number of the hardware bin that the tester assigned to the last part that was tested on this site. Returns -1 if the tester has not yet assigned a bin to the part or a value between 0 and 65535 if the bin was assigned.	Test Socket Threads Only Valid in PostUUT
HardwareBinNumbers	Array of Numbers	The numbers of the hardware bins that the tester assigned to the parts in the current batch.	Valid in PostBatch
IsDuplicateDieCoordinates	Boolean	Returns true if a part with the same die coordinates as the current part was tested previously in the current wafer.	Test Socket Threads Only Valid in MainSequence
IsDuplicatePartId	Boolean	Returns true if a part with the same part ID as the current part w	Test Socket Threads Only

		as tested previously in the current lot.	Valid in MainSequence
IsRetesting	Boolean	Returns true if the tester is currently retesting the same parts from the previous batch because the operator requested a retest.	Valid in PreBatch
IsStartOfWafer	Boolean	Returns true if the current batch is the first batch of parts of a new wafer. This property remains true if the first batch of parts are retested.	Valid in PreBatch
LotSummaryReportFilePath	String	Returns the absolute file path of the last Lot Summary Report generated for the current lot.	Valid in MainSequence
LotSummaryReportFilePaths	Array of strings	Returns the absolute file paths of all Lot Summary Reports generated for the current lot. If the Generate One File per Wafer option is enabled, one path is returned for each wafer processed in the lot so far. Otherwise, the length of the array is one.	Valid in MainSequence
NumberOfSites	Number	The number of sites that are testing the current lot.	—
PartId	String	Returns the part identifier for the current part being tested on this site. The handler/prober driver sets this p	Test Socket Threads Only Valid in MainSequence

		roperty value by setting values in the SitePartIds parameter in the StartOfTest handler driver entry point sequence. If the handler/prober driver does not set this property, the tester sets the value to a unique value if the GenerateUniquePartIds station settings is set to true. This property corresponds to the PART_ID field in the Part Results Record (PRR) in the STDF log file. When the `GenerateUniquePartIds` property is `True`, TSM reassigns the same unique Part ID to the part when it is retested. Customize the behavior of `GenerateUniquePartIds` to assign a new unique Part ID to a part when it is retested.	
PartText	String	Returns the part description text for the current part being tested on this site. The handler/prober driver sets this property value by setting values in the SitePartTexts parameter in the StartOfTest handler driver entry point sequence. This pro	Test Socket Threads Only Valid in MainSequence

		perty corresponds to the PART_TXT field in the Part Results Record (PRR) in the STDF log file.	
SiteNumber	Number	Site number of the current site.	Test Socket Threads Only
SiteNumbers	Array of Numbers	The site numbers that are testing the current lot.	—
SiteTestTimeInSeconds	Number	Returns the time to execute the `MainSequence` sequence for the last part that was tested on this site.	Test Socket Threads Only Valid in PostUUT
SoftwareBinName	String	The name of the software bin that the tester assigned to the last part that was tested on this site.	Test Socket Threads Only Valid in PostUUT
SoftwareBinNames	Array of Strings	The names of the software bins that the tester assigned to the parts in the current batch.	Valid in PostBatch
SoftwareBinNumber	Number	The number of the software bin that the tester assigned to the last part that was tested on this site. Returns -1 if the tester has not yet assigned a bin to the part or a value between 0 and 65535 if the bin was assigned.	Test Socket Threads Only Valid in PostUUT
SoftwareBinNumbers	Array of Numbers	The numbers of the software bins that the tester assigned to the parts in the current batch.	Valid in PostBatch

StdfLogFilePath	String	Returns the absolute file path of the last STDF log file generated for the current lot.	Valid in MainSequence
StdfLogFilePaths	Array of strings	Returns the absolute file paths of all STDF log files generated for the current lot. If the Generate One File per Wafer option is enabled, one path is returned for each wafer processed in the lot so far. Otherwise, the length of the array is one.	Valid in MainSequence
WaferRuntimeData.ExecDescription	String	Returns the wafer description supplied by exec. This property corresponds to the EXC_DESC field in the Wafer Results Record (WRR) in the STDF log file.	Valid in PreBatch
WaferRuntimeData.FabWaferId	String	Returns the fab wafer ID. This property corresponds to the FABWF_ID field in the Wafer Results Record (WRR) in the STDF log file.	Valid in PreBatch
WaferRuntimeData.FrameId	String	Returns the wafer frame ID. This property corresponds to the FRAME_ID field in the Wafer Results Record (WRR) in the STDF log file.	Valid in PreBatch
WaferRuntimeData.MaskId	String	Returns the wafer mask ID. This property corresponds to the MASK_ID field in the Waf	Valid in PreBatch

		er Results Record (WRR) in the STDF log file.	
WaferRuntimeData.UserDescription	String	Returns the wafer description supplied by user. This property corresponds to the USR_DESC field in the Wafer Results Record (WRR) in the STDF log file.	Valid in PreBatch
WaferRuntimeData.WaferId	String	Returns the wafer ID. This property corresponds to the WAFER_ID field in the Wafer Information Record (WIR) and Wafer Results Record (WRR) in the STDF log file.	Valid in PreBatch

Access Restrictions

Some properties are valid only when accessed from certain process model threads. If you access these properties from an unsupported model thread, the step reports a run-time error.

Some properties are valid only at certain times during execution. If you access these properties before they are valid, the step either returns default values or reports a run-time error as described below.

- **Test Socket Threads Only**—You must access these site-specific properties from a test socket thread. The step reports a run-time error if you access the property in a sequence that executes in a controller thread, such as `ProcessSetup` when executing the Batch process model. You can access these properties from any sequence when using the Sequential process model.
- **Controller Thread Only**—You must access these properties from a process model callback sequence that executes in a controller thread, such as `ProcessCleanup`. The step reports a run-time error if you access the property in a sequence that executes in a test socket thread, such as `MainSequence` or `PreUUT`.

- **Valid in PreBatch**—TSM assigns values to these properties before testing starts for the current batch. These properties have meaningful values only in the following locations. The step returns default values if you access the property from other locations. The step returns default values if you access the property from other locations.
  - In the Model Plugin – Pre Batch entry point and beyond in model plug-ins when using the Batch process model.
  - In the Model Plugin – Pre UUT entry point and beyond in model plug-ins when using the Sequential process model.
  - In the PreUUT Model callback and beyond in test programs.
- **Valid in MainSequence**—TSM assigns values to these site-specific properties before testing starts for the current part. These properties have meaningful values only in the following locations. The step returns default values if you access the property from other locations.
  - In the Model Plugin – Pre UUT entry point and beyond in model plug-ins.
  - In the `MainSequence` sequence and beyond in test programs.
- **Valid in Post UUT**—TSM assigns values to site-specific properties after testing completes for the current part. These properties have meaningful values only in the following locations. The step reports a run-time error if you access these properties from other locations.
  - In the Model Plugin – UUT Done entry point and beyond in model plug-ins.
  - In the PostUUT Model callback and beyond in test programs.
- **Valid in Post Batch**—TSM assigns values to these properties after testing completes for all sites. These properties have meaningful values only in the following locations. The step reports a run-time error if you access these properties from other locations.
  - In the Model Plugin – Batch Done entry point and beyond in model plug-ins when using the Batch process model.

- In the Model Plugin – UUT Done entry point and beyond in model plug-ins when using the Sequential process model.
- In the PostBatch callback and beyond in test programs when using the Batch process model.
- In the PostUUT callback and beyond in test programs when using the Sequential process model.

▾ Custom Test Conditions

Access the custom test conditions you created in the Test Program Editor.

- **Name**—The name of the lot setting, station setting, STS state, execution data, or custom test condition to access. You can choose one of the items from the drop-down menu, or you can type any part of the item name and select the item from the drop-down menu of suggestions.
- **Destination Expression**—An expression that specifies the location to copy the value. You typically specify a local variable that you can use throughout the sequence.

Control STS Test Head Step (TSM)

Use the Control STS Test Head step to control properties of the STS. The step must be run only on an STS with STS Maintenance Software 19.0 or later.

Configuring the Step

Use the Control STS Test Head edit tab in the TestStand Sequence Editor to specify the list of STS properties and settings.

Control STS Test Head Tab (TSM)

The Control STS Test Head tab contains a table that lists the STS properties the step controls.

Modifying the Layout and Entering Data

Use the buttons located to the right of the table to add, remove, or reorder per-site inputs.

You can enter data in the table in the following ways:

- Enter data when a cell is highlighted.
- Click in a cell when it is highlighted.
- Double-click a cell.
- Select a value from a drop-down menu.
- Drag a variable or property from the Variables pane to a text or expression cell.

Columns

The table contains the following columns:

- **Name**—The name of the STS property to control. You can choose one of the items from the drop-down menu, or you can type any part of the property name and select the item from the drop-down menu of suggestions.
- **Source Expression**—An expression that specifies the value to set for the STS property.

The following table lists the STS properties you can access from this step.

Name	Type	Description
12VPowerSupplyEnabled	Boolean	Enables or disables the 12 V pins on STS CX and STS DX systems. The pins are disabled by default at STS power up and when a load board is not installed or locked down on the STS, such as when the value of the `DIBPresent` property is `False`. These signals are available on the system cable for STS CX systems and

		on the P143 system block for STS DX systems.
48VPowerSupplyEnabled	Boolean	Enables or disables the 48 V pins on STS CX and STS DX systems. The pins are disabled by default at STS power up and when a load board is not installed or locked down on the STS, such as when the value of the `DIBPresent` property is `False`. These signals are available on the system cable for STS CX systems and on the P143 system block for STS DX systems.

Inline QA Test Block Step (TSM)

Use the Inline QA Test Block step to insert in the `MainSequence` sequence of the test program main sequence file a block of Inline QA Test Block and End steps, in which you can insert additional steps that call code modules that perform the inline QA tests. The steps within the block execute only when all of the following conditions are true:

- The StationSettings.Standard.InlineQAEnabled Boolean property is `True`.
- The `Step.ConditionExpr` property of the Inline QA Test Block step specifies an expression that evaluates to `True`.
- The next inline QA state, which is the next Boolean value removed from the TSM queue of inline QA states, is `True`.

Notes

- If multiple inline QA test blocks exist in the sequence, only the first one that executes dequeues the inline QA state for the DUT. Subsequent inline QA test blocks use the inline QA state that the first inline QA test block dequeued.
- The default expression for the `Step.ConditionExpr` property evaluates to `T`

rue when the MainSequence sequence has not yet encountered a sequence failure. You can modify the expression the Step.ConditionExpr property specifies to customize the condition for which the inline QA test block executes.

- You can place multiple Inline QA Test Block step type instances at any location in any test sequence, but NI recommends that you place only a single instance after all standard test steps in the MainSequence sequence of the test program main sequence file.

Set and Lock Bin Step (TSM)

Use the Set and Lock Bin step to use an expression to assign a software bin to a DUT and override the TSM automatic bin assignment.

Use the **Bin Expression** control on the Set and Lock Bin tab to specify an expression that evaluates at run time to a valid software bin number defined in the bin definitions file. The step assigns the software bin to the DUT and locks the bin. Once the bin is locked, subsequent failing tests on Semiconductor Multi Test steps report a run-time error.

Note The Set and Lock Bin step assigns the software bin to the DUT regardless of whether the test program previously assigned a different software bin to the DUT. An instance of the Set and Lock Bin step can override a bin assigned by a previous Set and Lock Bin step. Therefore, NI recommends that you use a Set and Lock Bin step only after all tests have executed.

You can use the Set and Lock Bin step to implement grading in a test program.

Set and Lock Bin Tab (TSM)

The Set and Lock Bin tab contains the following options:

- **Bin Expression**—An expression that evaluates at run time to a valid software bin number defined in the bin definitions file associated with the test program. You can use the drop-down menu to select a software bin defined in the bin definitions file.

See Also

Binning DUTs Based on Test Results

Grading Passed DUTs

Load Correlation Offsets Step (TSM)

Use the Load Correlation Offsets step only in the `ProcessSetup` sequence to load and apply correlation offset values to test results on a per-site basis. The step applies correlation offset values at run time before evaluating the test result data against limits. The step stores the offset value to apply in the `Step.Result.Evaluations[index].NumericLimit.CorrelationOffset` property on Semiconductor Multi Test steps. The Semiconductor Multi Test step adds the correlation offset to the test result data before evaluating against limits. The Debug Test Results Log and CSV Test Results Log include the correlation offset values. The STDF Log includes the correlation offsets file path as a DTR with an `NIDTR:` prefix to record the path of the correlation offsets file in a `<CorrelationOffsetsFilePath>` XML tag. For example, if you specify `C:\Data\TestPrograms\mypath.txt` in the **Correlation Offsets File Path** option of the Load Correlation Offsets step edit tab, the STDF Log includes the following information:

```
NIDTR:<CorrelationOffsetsFilePath>C:\Data\TestPrograms\mypath.txt</CorrelationOffsetsFilePath>
```

If the length of the text to add is more than 255 characters, TSM splits the text among multiple DTRs, each with an `NIDTR:` prefix.

When this step is used to load a correlation offsets file, it stores the file path in the `CorrelationOffsetsFileAbsolutePath` lot settings property. To access this file path or determine whether correlation offsets have been loaded at runtime, use the Get Test Information step to read the value of this property.

Configuring the Step

Use the Load Correlation Offsets Step edit tab in the TestStand Sequence Editor to specify the correlation offsets file to use.

Load Correlation Offsets Tab (TSM)

The Load Correlation Offsets tab contains the following options:

- **Correlation Offsets File Path**—Path to the correlation offsets file to use. TSM sets the CorrelationOffsetsFileAbsolutePath property at run time with this value.
- **Specify by Expression**—Enable this option to use an expression to specify the path to the correlation offsets file to use.

Perform Part Average Testing Step (TSM)

Use the Perform Part Average Testing step to perform part average testing for any tests that have part average testing enabled and have already executed for the current DUT. Specify the Perform Part Average Testing step settings in the PAT Algorithm Settings panel of the Test Program Editor. Tests performed by the Perform Part Average Testing step will not be performed after the `MainSequence` sequence of the test program executes.

The step sets the `Result.Status` property based on the result of the tests that it performs.

Status	Test Results
Done	The step did not execute any tests.
Passed	The step executed at least one test and all tests passed.
Failed	The step executed at least one test and at least one test failed.
Error	A run-time error occurred when executing the P AT algorithm.

Configuring the Step

To enable the step, set the value of the `PartAverageTestingAlgorithmDescription.EnvironmentSettings.EnablePerformPartAverageTestingStep` file global variable to `True` in the PAT callback sequence file. TSM reports a run-time error if it executes a Perform Part Average Testing step and this step type is not enabled.

You do not need to configure the Step Settings pane for the step to perform part average tests. The step displays the test results under the Tests tab of the Step Settings pane during program execution, after the step executes. Once the test program finishes executing, the Tests tab is empty.

Configure Lot Settings Dialog Box (TSM)

Select **Semiconductor Module»Configure Lot** in the TestStand Sequence Editor, click the **Configure Lot** button on the TSM toolbar, or click the **Configure Lot** button in the default TSM operator interface to launch the Configure Lot Settings dialog box, in which you can specify settings for the current lot under test.

Note If an active test configuration you select for a test program contains any of the following fields, TSM dims the field in this dialog box because the test program obtains the value from the test configuration at run time.

The Configure Lot Settings dialog box contains the following options:

- **Test Program Path**—When the Test Program Directory option on the General tab of the Configure Station Settings dialog box is empty, use this option to browse to a file path. If you specify a value for the Test Program Directory option, this control lists the sequence files in the directory you specified.
- **Test Program Configuration**—Specifies the test program configuration to execute. This option is available only when you define configurations for a test program.
- **Part Type**—Populates the `LotSettings.Standard.PartType` property, which specifies the PART_TYP field in an STDF MIR record.

- **Lot ID**—Populates the `LotSettings.Standard.LotId` property, which specifies the LOT_ID field in an STDF MIR record.
- **Estimated Lot Size**—Populates the `LotSettings.Standard.LotSize` property, which an inline QA algorithm can use to determine for which DUTs to enable inline QA.
- **Test Flow**—Populates the `LotSettings.Standard.TestFlowId` property, which specifies the FLOW_ID field in an STDF MIR record.
- **Test Temperature**—Populates the `LotSettings.Standard.TestTemperature` property, which specifies the TST_TMP field in an STDF MIR record.
- **Operator Name**—Displays the value of the `LotSettings.Standard.OperatorName` property, if it exists. If this property is blank, this field displays the user name of the currently logged in user. TSM uses this value to specify the OPER_NAM field in an STDF MIR record.
- **Enabled Sites**—Populates the `StationSettings.Standard.AvailableSiteNumbers` property, which specifies which site numbers to use when running a test program. TSM obtains the sites to display in this list from the pin map for the test program or from the number of test sockets specified in the Model Options dialog box if the test program does not use a pin map. You must enable at least one site. If the test program uses the Sequential process model, you can enable only one site.

See Also

Customizing the Behavior for Obtaining Lot Settings

Configure Built-in Simulated Handler Dialog Box (TSM)

The Configure Built-in Simulated Handler dialog box contains the following options:

- **Number of DUTs to Test**—Specifies the number of DUTs to test before the simulated handler driver notifies the tester to stop testing. When you start testing by clicking the **Start Lot** button on the TSM toolbar, the simulated handler driver stores an internal count of the number of DUTs that have completed testing.

- **Simulated Index Time**—Specifies the amount of time, in milliseconds, to elapse before the simulated handler driver notifies the tester to begin the next iteration of testing. When you start testing by clicking the **Start Lot** button on the TSM toolbar, the simulated handler driver waits the specified simulated index time from the end of the EndOfTest simulated handler driver entry point before sending the start-of-test notification, which allows the StartOfTest simulated handler driver entry point to begin the next iteration of testing. When you execute a test using the Single Pass Execution entry point, the simulated handler driver ignores any value specified in this control and performs only a single test iteration.
- **Show Handler Dialogs**—When you enable this option, the simulated handler driver launches a dialog box when invoking the StartOfTest and EndOfTest simulated handler driver entry points.

See Also

Configure Handler Driver Entry Point

Configure Station Settings Dialog Box (TSM)

Select **Semiconductor Module»Configure Station** in the TestStand Sequence Editor, click the **Configure Station** button on the TSM toolbar, or click the **Configure Station** button in the TSM operator interface to launch the Configure Station Settings dialog box, in which you can specify settings for the current test station.

The Configure Station Settings dialog box contains the following tabs:

- General—Configures settings for the handler/prober mode, Inline QA, and step failure mode.
- Advanced—Contains buttons to launch the Result Processing, Station Options, Edit Search Directories, and LabVIEW Adapter Configuration dialog boxes.

See Also

Customizing the Behavior for Obtaining Station Settings

Configure Station Settings Dialog Box (TSM)

Advanced Tab

The Advanced tab contains the following options:

- **Result Processing**—Launches the Result Processing dialog box, in which you specify and configure the result processors to execute during test program execution.
- **Station Options**—Launches the Station Options dialog box, in which you configure advanced options for the test station.
- **Search Directories**—Launches the Edit Search Directories dialog box, in which you configure the search directories to use to find files during execution.
- **LabVIEW Adapter**—Launches the LabVIEW Adapter Configuration dialog box, in which you specify options for calling LabVIEW code. Select the **Development System** option for debugging. Select the **LabVIEW Run-Time Engine** option for execution.

See Also

Configure Station Settings dialog box

Configure Station Settings Dialog Box (TSM)

General Tab

The General tab contains the following options:

- **Enable Handler/Prober Driver (Real or Simulated)**—Specifies handler/prober options.

- **Handler/Prober Driver**—Specifies the handler/prober driver to use. The drop-down menu lists the Built-in Simulated Handler Driver first and any other installed drivers in alphabetical order. Select the **Built-in Simulated Handler Driver** option to use the NI Built-in Simulated Handler Driver to simulate handler functions when you do not have access to a real handler.
- **Configure Handler/Prober**—Launches a dialog box, which you use to specify handler options.

- **Enable Inline QA**—Specifies whether to enable inline QA. When you enable this option, specify a sequence file to determine the frequency of inline QA in the **Inline QA Algorithm** control. The drop-down menu displays only algorithms located in the `<TestStand Public>\Components\Modules\NI_SemiconductorModule\InlineQA` directory.
- **Test Program Directory**—(Optional) Limits the selection of the test program to the directory you specify.
- **Step Failure Mode**—Specifies how the test program continues after a failure occurs. This option mimics the functionality of the **Immediately Goto Cleanup On Sequence Failure** option on the Execution tab of the Station Options dialog box.

  - **Continue on Step Failure**—Specifies to continue testing after a failure occurs.
  - **Goto Cleanup on Step Failure**—Specifies to stop testing after a failure occurs.

- **Site Status Part Count**—Specifies the number of part test results tracked per site. The site status indicators in the operator interface display the results of the last **n** parts, where this option specifies **n**.

See Also

Configure Station Settings dialog box

Create Test Program from Digital Pattern Project Dialog Box (TSM)

Select **Semiconductor Module»Create Test Program from Digital Pattern Project** to launch the Create Test Program from Digital Pattern Project dialog box. Use this dialog box to create a basic test program with a structure for initializing sessions for NI-Digital and NI-DCPower instruments and bursting the patterns defined in the digital pattern project. Use this basic test program as a starting point to build a more comprehensive test program.

The Create Test Program from Digital Pattern Project dialog box contains the following options:

- **Source Digital Pattern Project File Path**—Specifies the digital pattern project file from which you want to create a test program.
- **Target Test Program Directory Path**—Specifies the directory in which you want to create the test program from the digital pattern project.
- **Errors and warnings (optional output)**—Displays errors and warnings. You must resolve errors before you can create the test program.

The generated test program for a digital pattern project consists of the following components:

- **Sequence File**—Implements the test program for the digital pattern project. The test program defines the three sequences described below. For each sequence, replace the placeholder labels with an implementation from your measurement library or a step template provided by another add-on.
  - **ProcessSetup**—Use this sequence to initialize sessions for NI-Digital and NI-DCPower instruments defined in the active pin map file of the digital pattern project. Sessions for NI-Digital should use the active pin levels sheet and the active timing sheet of the digital pattern project.
  - **MainSequence**—Use this sequence to burst the patterns defined in the digital pattern project and optionally capture waveforms.
  - **ProcessCleanup**—Use this sequence to close sessions for NI-Digital and NI-DCPower instruments defined in the active pin map file of the digital pattern project.

- **Supporting Materials**—Contains the following files and subdirectories:
  - A copy of the source digital pattern project file
  - **Bin Definitions**—Contains the default bin definitions.
  - **Pin Maps**—Contains copies of pin map files from the source digital pattern project.
  - **Specifications**—Contains specification files.
  - **Digital**—Contains copies of data files for digital tests from the source digital pattern project, including the following files:
    - Pattern files
    - Timing files
    - Levels files
    - Source and capture waveform files
  - **Offline Mode Configurations**—Defines an Offline Mode System Configuration that includes the NI-Digital and NI-DCPower instruments in the pin map file. Use this file to simulate the instruments needed by the test program in Offline Mode.
    Ensure instruments in the pin map file follow the recommended instrument naming convention for semiconductor test programs:

    InstrumentType_ModelNumber_PXIChassisLocation_SlotLocation, for example, `HSD_657x_C2_S03`, where `InstrumentType` is an ASCII description of the instrument, `ModelNumber` is the model number as defined on `ni.com`, `PXIChassisLocation` uses a single digit to identify the PXI chassis (`Cx`), and `SlotLocation` uses double digits to identify the slot location (`Sxx`).

To find the digital pattern project for the test program, select **Semiconductor Module»Edit Test Program...** then select Digital Pattern Project.

CSV Test Results Log Options Dialog Box (TSM)

Enable and configure the TSM result processing plug-ins to launch the CSV Test Results Log Options dialog box, in which you can specify settings for the CSV Test Results Log.

The CSV Test Results Log Options dialog box contains the following options:

- **CSV Log Destination Directory**—Absolute path of the directory in which you want TSM to create the CSV data log file. Leave the control blank if you want TSM to create the CSV data log file in the same directory as the test program main sequence file. During testing, the CSV Test Results Log result processor writes data to a temporary file with an extension of `.csvtemp` in this directory at the end of each batch. When the file completes, the CSV Test Results Log result processor renames the file to the final report filename you specify.
- **Generate One File per Wafer**—Specifies to create a separate CSV log file for each tested wafer. This option has no effect when testing without a wafer probe. Each new log file resets the batch number to `1`.
- **Log Wafer Data**—Specifies to create columns in the CSV log file for the following data:
  - Wafer ID
  - Die X Coordinate
  - Die Y Coordinate
- **Log Code Module Execution Time**—Specifies to create a column in the CSV log file for the code module execution time for each test.
- **Limit Number of Test Data Records**—Specifies to limit the number of individual part test records in the CSV log file to one of out every **N** DUTs you specify per site. When you enable the **Generate One File per Wafer** option, this option applies to each CSV file independently and not to the entire lot. Each wafer CSV file includes test records for one out of every **N** DUTs per site.

End of Test Dialog Box (TSM)

The NI Built-in Simulated Handler Driver launches the End of Test dialog box to display hardware and software bin information during the EndOfTest entry point when you enable the Show Handler Dialogs option on the Configure Built-in Simulated Handler Driver dialog box and you use the Batch or Sequential process model.

The End of Test dialog box contains the following elements:

- **Site Bins**—Displays the hardware and software bins that correspond to the DUT tested at each site on a per-test basis.
- **Bin Totals**—Displays the total number of DUTs per hardware and software bin on a per-lot basis.
- **Show/Hide Details**—Shows or hides the Bin Totals table.
- **Do not show this dialog again for this lot**—Enable this option to hide the dialog box for the remainder of lot testing.

See Also

NI Built-in Simulated Handler Driver

Log Browser Window

Click the **Log Browser** button near the upper right corner of the Test Program Performance Analyzer to open the Log Browser window, which you can use to view or compare multiple log files that represent different modifications of a test program.

Complete the following steps to view or compare log files.

1. In the **Log Directory** path control, select the top-level directory that contains the log files you want to view to compare. The column on the left displays relative subdirectories.
2. Complete the following steps to view file(s) in Single Data Set mode or Compare Data Sets mode.

a. **Single Data Set** – Double-click the row of the file you want to load. The background row color turns blue.

b. **Compare Data Sets** – Right-click the row of the base log file you want to load and chose **Select Base Log File** from the context menu. The background row color of the base log file you selected turns gray. Right-click the row of the modified log file you want to load and chose **Select Modified Log File** from the context menu. The background row color of the modified log file you selected turns blue.

Lot Statistics Viewer (TSM)

Select **Semiconductor Module»Show Lot Statistics Viewer** or click the **Show Lot Statistics Viewer** button on the TSM toolbar to launch the Lot Statistics Viewer window, in which you can view lot statistics, including per-site bin counts, while running or debugging a sequence in the sequence editor. You can also control test program execution in the Lot Statistics Viewer.

The Lot Statistics Viewer displays a new tab for each test program sequence file you execute. The tab includes a table of the software bin statistics for each site and highlights the cell for each updated DUT result. When execution completes, the table dims. Click the expand/collapse button to the left of the software bin name to collapse the view to show only the total part count or to expand the view to show part counts for each software bin.

When you use the `AvailableSiteNumbers` property on the NI_SemiconductorModule_StationSettings data type to specify which site numbers from a pin map for a test program to use when running the test program, the Lot Statistics Viewer displays only the sites you specify.

The Lot Statistics Viewer window includes the following options:

- **Configuration**—Configures the lot with the selected configuration for the test program that corresponds to the active sequence file. This control is disabled if the active sequence file has no configurations.
- **Single Test**—Starts a lot and tests a single DUT for each site for the active sequence file if no lot is active and pauses when complete or, if paused, tests a single DUT for each site before pausing again.

- **Start/Resume Lot**—Starts testing a new lot, resumes a suspended sequence execution at a breakpoint, or resumes a sequence execution that is paused between DUTs.
- **Pause**—Pauses testing of the lot. Testing pauses between DUTs after completing the tests for the current DUTs on each site and before TSM sends the end-of-test (EOT) signal to the handler or prober.
- **Retest**—After a single test completes or when you pause a lot, retests a single DUT for each site for the active sequence file and then pauses execution. The lot statistics update to include only the results from the last retest for a given DUT.
- **End Lot**—Ends testing the current lot. Click this button instead of selecting the **Debug»Terminate All** menu item to safely end a lot.
- **Hide Empty Bins**—Removes the empty bins from the display. This option is enabled by default.

Lot Summary Options Dialog Box (TSM)

Enable and configure the TSM result processing plug-ins to launch the Lot Summary Options dialog box, in which you can specify settings for the STDF Log file.

The Lot Summary Options dialog box contains the following options:

- **Lot Summary Report Destination Directory**—Absolute path of the directory in which you want TSM to create the report file. Leave the control blank if you want TSM to create the report file in the same directory as the test program main sequence file.
- **Generate One File per Wafer**—Specifies to create a separate Lot Summary Report file for each tested wafer. This option has no effect when testing without a wafer probe.

Viewing TSM Data at Runtime

Use the TSM Runtime Data Viewer to see test results and debug issues at runtime.

1. To start logging, open the Runtime Data Viewer from **Semiconductor Module»Show Runtime Data Viewer**.

2. Click between the following tabs to see your data in different formats.
   - **High Level Grid**– Each row represents a test and each column represents a part. TSM adds new parts on the left as they are tested.
     TSM shows the test result value for numeric tests and a blank cell for pass / fail tests. For each result, a green colored cell indicates a passed test while a red colored cell indicates a failed test.
   - **Verbose View**– TSM shows each test on each part in the order that they are executed.
3. (Optional) Filter the data you see. Refer to **Filtering Runtime Data in the High Level Grid** and **Filtering Data in the Verbose View** for more information.
4. (Optional) To copy the filtered data, click **Copy filtered list to clipboard** .
5. (Optional) To clear all results, click **Clear all test results** . TSM will also clear all results when starting a new lot.
6. To stop logging, close the Runtime Data Viewer.

Filtering Runtime Data in the High Level Grid

Filter the data you see in the High Level Grid tab of the Runtime Data Viewer.

In the Filter for field, specify the category you want to filter on and the value you want to filter for. Use the syntax, `category:value`, when creating a filter. Separate multiple filter requirements with a comma. For example, `site:0, test #:100-102, partId:1`.

Use the following categories to filter your data.

- site
- test#
- batch
- partId
- step or stepName
- test or testName

- status
- X or dieCoordinateX
- Y or dieCoordinateY

Filtering Data in the Verbose View

Filter the data you see in the Verbose View tab of the Runtime Data Viewer.

To filter, click the filter button on any column and specify the value you want to filter for.

To reveal or hide a column, click **Select columns and rows to view** and enable or disable the columns. You can also limit the number of results to the last `N` batches.

Semiconductor Module Run-Time Error Dialog Box

The Semiconductor Module Run-Time Error dialog box launches when a run-time error occurs in an execution that uses a process model if TSM is enabled. The dialog box contains a description of the error and the step, sequence, and sequence file where the error occurred. The dialog box includes an error code only for TestStand errors unrelated to TSM. Use the **More Options** and **Fewer Options** buttons to show or hide the controls that specify how to handle the run-time error.

The Semiconductor Module Run-Time Error dialog box contains the following run-time error handling options:

- **End lot after running cleanup**—Execution proceeds to the Cleanup step group for the sequence, and lot testing ends.
- **Run cleanup and continue testing**—Execution proceeds to the Cleanup step group for the sequence, and lot testing continues with the next DUT.
- **Retry**—TestStand executes the step again.
- **Ignore**—TestStand does not set the status of the sequence to `Error`. Instead, TestStand sets the `Error.Occurred` property of the step to `False`, and execution continues normally with the next step in the sequence. The `Result.Status` property of the step remains set to `Error`.

- **Abort immediately**—TestStand stops execution immediately without running any Cleanup steps.

The text on the following buttons change depending on the option you select for handling the run-time error:

- **Break, Then Option**—TestStand suspends execution at the step that caused the run-time error. If you resume the execution after closing the dialog box, TestStand performs the actions for the selected run-time error handling option as described above. This button is hidden when you select the **Abort immediately** option.
- **Option**—TestStand performs the actions for the selected run-time error handling option as described above.

The Semiconductor Module Run-Time Error dialog box also provides the following options that change how TestStand handles subsequent run-time errors:

- **Do this for all Run-Time Errors in this Execution**—Performs the selected run-time error handling option for any run-time errors that occur later in the execution without displaying the Semiconductor Module Run-Time Error dialog box.
- **Do this for all Run-Time Errors in this Batch**—Performs the selected run-time error handling option for any run-time errors that occur later in any execution that is associated with the current batch without displaying the Semiconductor Module Run-Time Error dialog box. This option is available only when you use the Batch process model.

Start of Test Dialog Box (TSM)

The NI Built-in Simulated Handler Driver launches the Start of Test dialog box during the StartOfTest entry point when you enable the Show Handler Dialogs option on the Configure Built-in Simulated Handler Driver dialog box and you use the Batch or Sequential process model. The Start of Test dialog box displays information about the number of DUTs tested, the number of DUTs left to test, and the active sites.

When using the Batch process model, use the checkboxes to enable or disable sites. Enable the **Do not show this dialog again for this lot** option to hide the dialog box for the remainder of lot testing.

See Also

NI Built-in Simulated Handler Driver

STDF Log Options Dialog Box (TSM)

Enable and configure the TSM result processing plug-ins to launch the STDF Log Options dialog box, in which you can specify settings for the STDF Log file.

The STDF Log Options dialog box contains the following options:

- **STDF Log Destination Directory**—Absolute path of the directory in which you want TSM to create the data log file. Leave the control blank if you want TSM to create the data log file in the same directory as the test program main sequence file. During testing, the STDF Log result processor writes data to a temporary file with an extension of `.stdtemp` in this directory at the end of each batch. When the file completes, the STDF Log result processor renames the file to the final report filename you specify.
- **Generate One File per Wafer**—Specifies to create a separate STDF log file for each tested wafer. This option has no effect when testing without a wafer probe.
- **Limit Number of Test Data Records**—Specifies to limit the number of individual part test records in the STDF log to one of out every **N** DUTs you specify per site. Individual test records include PTR, FTR, and DTR. The summary test result records include the test result records of the parts for which you omitted individual test records. When you enable the Generate One File per Wafer option, this option applies to each STDF file independently and not to the entire lot. Each wafer STDF file includes test records for one out of every **N** DUTs per site.

Test Program Performance Measurement Configuration Dialog Box (TSM)

Select **Semiconductor Module»Measure Performance of <filename>** in the TestStand Sequence Editor to launch the Test Program Performance Measurement Configuration dialog box, in which you can specify settings for measuring test program performance. The changes you make persist in the dialog box.

The Test Program Performance Measurement Configuration dialog box contains the following options:

- **Test Program**—Displays the test program on which to measure performance.
- **Process Model**—Displays the process model the test program uses.
- **Operator Interface Path**—By default, specifies the absolute path to the default LabVIEW operator interface TSM installs. Use the browse button to navigate to a custom operator interface to use instead of the default TSM LabVIEW operator interface or paste the path to the executable into the control. This control cannot be empty.
- **Output File Directory**—Specifies the directory in which to write the log files. Use the browse button to navigate to a custom output directory or paste the path to the directory into the control. The path must be an absolute path.
- **Number of Parts Per Site**—Specifies the number of parts to run on each site.
- **Site Configurations**—Each row corresponds to a lot and contains a comma-separated list of integers or integer ranges that specifies the sites the lot contains, for example: 0, 1, 2-7. The sites must exist in the pin map. Use the **Add Site Configuration** and **Delete Site Configuration** buttons to add and delete rows. TSM ignores empty rows at run time.

Note If you are using the Sequential process model, **Site Configurations** displays a single configuration that contains a single site (0). If you switch back to using the Batch process model, **Site Configurations** displays the previous set of configurations.

- **Configure Lot Settings**—Launches the Configure Lot Settings dialog box. If you have not configured lot settings for the active test program, TSM disables the OK button and displays a warning to prompt you to configure the lot settings.

Click the expand/collapse button to view or hide the **Advanced** section of the dialog box, which includes the following option that configures the testing environment to simulate a production environment:

- **Operator Interface Command Line**—The default command line is `/NoCompatibilityIssuesDialog /run "Measure Test Program Performance" "<TestStand>\Components\Modules\NI_SemiconductorModule\NI_SemiconductorModule_PerformanceMeasurement.seq" /quit`, which configures the execution environment. If you use a custom sequence file instead of the default `NI_SemiconductorModule_PerformanceMeasurement.seq` file to configure the environment for performance testing, modify the command line to call the custom sequence file.

Note If you use an operator interface with a custom command-line parser, modify the command-line arguments to ensure that the operator interface calls the `Measure Test Program Performance` sequence of the `NI_SemiconductorModule_ PerformanceMeasurement.seq` file located in the `<TestStand>\Components\Modules\NI_SemiconductorModule` directory.

- **Reset to Default**—Reverts the value of the **Operator Interface Command Line** option to use the default command-line value.
- **Allow execution with LabVIEW Development Environment Adapter**—By default, when you select the LabVIEW Development Environment adapter in TestStand, the adapter temporarily switches to use the LabVIEW Run-Time Engine to simulate production performance. Enabling this option disables the switch to the LabVIEW Run-Time Engine, allowing performance measurement with the LabVIEW Development Environment adapter. This

increases test time. This option is disabled if the LabVIEW adapter is already set to use the LabVIEW Run-Time Engine.

Note Changing any option under **Advanced** causes the section to remain expanded until the next time the dialog box is opened.

Debug Test Results Log Options Dialog Box (TSM)

Enable and configure the TSM result processing plug-ins to launch the Debug Test Results Log Options dialog box, in which you can specify settings for the Debug Test Results Log.

The Debug Test Results Log Options dialog box contains the following options:

- **Debug Test Results Log Destination Directory**—Absolute path of the directory in which you want TSM to create the data log file. Leave the control blank if you want TSM to create the data log file in the same directory as the test program main sequence file.
- **Report Orientation**—Specifies the orientation of the Debug Test Results Log. The default is portrait orientation. Landscape orientation uses wider columns for tests with long test numbers or test names. Landscape orientation does not maintain report column alignment when test names exceed 101 characters.
- **Log Results Only for DUT Failures**—Includes results in the Debug Test Results Log only if the DUT fails testing, as determined by the bin assignment for the DUT. If a DUT passes testing, TSM does not update the Debug Test Results Log.
- **Limit Number of Results Displayed in Report View**—Limits the number of results to display in the report views in the TSM Operator Interface and TestStand Sequence Editor to the number of DUTs you specify.

TSM Application API

The TSM Application API is a component that simplifies custom operator interfaces, custom test reports and logs, and other applications for controlling or monitoring a test system. The ConfigureLotSettings and ConfigureStationSettings callback

sequences also use the TSM Application API to obtain information about the test system.

Using the TSM Application API in LabVIEW Applications

TSM does not provide a full set of VIs for every TSM Application API method and property. You can access the TSM Application API COM server using the LabVIEW ActiveX Invoke and Property nodes to access the methods and properties. The Application Development palette in LabVIEW includes a set of wrapper VIs that use the TSM Application API COM server for some functionality. To access the wrapper VIs, add the `SemiconductorModuleManager.mnu` palette from the `<LabVIEW>\vi.lib\NI_TestStand_SemiconductorModule\NI_TestStand_Semiconductor_Module.lvlibp` project library to the palette in LabVIEW. Refer to the **LabVIEW Help** for more information about customizing palettes. In LabVIEW, select **Help»LabVIEW Help** to launch the **LabVIEW Help**.

Using the TSM Application API in .NET Applications

Use the TSM Application API .NET class library to access the TSM Application API. Add a reference to the `<TestStand>\API\DotNET\Assemblies\CurrentVersion\NationalInstruments.TestStand.SemiconductorModule.ApplicationAPI.dll` assembly to your Visual Studio project.

See Also

Creating the Semiconductor Module Manager Object

Programming with TSM APIs in C#

Using the Semiconductor Module Manager Object

Creating or Obtaining the Semiconductor Module Manager Object

Applications that use the TSM Application API must first create or obtain an instance of the Semiconductor Module Manager object, which implements most of the methods in the TSM Application API.

For applications that control a test system, such as an operator interface, you must create a new Semiconductor Module Manager object. For custom reports, callbacks

used to set lot of station settings, and other programs for monitoring a test system, you must obtain the existing Semiconductor Module Manager object that the TestStand Sequence Editor or an operator interface created.

Creating an Instance of the Semiconductor Module Manager

Before you create the Semiconductor Module Manager object, you must first create a TestStand Application Manager control and a SequenceFileView Manager control in the TestStand user interface.

To create the Semiconductor Module Manager object in LabVIEW, use the Create Semiconductor Module Manager VI. To create the object in .NET, call the `NewSemiconductorModuleManager` static method on the `SemiconductorModuleManagerFactory` class.

Obtaining an Existing Instance of the Semiconductor Module Manager

To obtain the existing Semiconductor Module Manager object in LabVIEW, use the Get Semiconductor Module Manager VI. To obtain the existing object in .NET, call the `GetSemiconductorModuleManager` static method on the `SemiconductorModuleManagerFactory` class.

See Also

Using the Semiconductor Module Manager Object

Using the Semiconductor Module Manager Object

To control the test system programmatically, call methods directly on the Semiconductor Module Manager object, such as StartLot, PauseLot, and EndLot.

Controlling Test Systems Using Buttons on Operator Interfaces

Complete the following tasks to use the Command objects to control the test system using buttons on an operator interface.

- Get a specific Command object that corresponds to the action for the button on the operator interface, such as StartLot or EndLot, by calling the `GetCommand` method on the Semiconductor Module Manager object.
- Use the properties on the Command object to determine the text to display and the enabled state of the button.
- Use the events on the Command object to determine when to update the text or enabled state of the button.
- Use the `Execute` method on the Command object to initiate the action.

Monitoring the State of the Test System

Complete the following tasks to use an Observer object to monitor the state of the test system.

- Create an Observer object by calling the `CreateObserver` method on the Semiconductor Module Manager object.
- Use the events on the Observer object to determine when changes to test system state occur.
- Use the `TestingState` property on the Semiconductor Module Manager object to determine the current state of the test system.
- Use the `TesterStatus` property on the Semiconductor Module Manager object to get a description of the current state of the test system.

Obtaining Test Statistics

Complete the following tasks to use the Lot Statistics object to obtain test statistics for individual test sites or for all test sites. The statistics are available for the entire lot or for the current wafer in the lot.

- Get the Lot Statistics object for an individual test site by calling the `GetSiteLotStatistics` or `GetWaferSiteLotStatistics` methods on the Semiconductor Module Manager object.
- Get the Lot Statistics object for all test sites by accessing the `AllSiteLotStatistics` or `WaferAllSiteLotStatistics` properties on the Semiconductor Module Manager object.

- Use the properties and methods on the Lot Statistics object to get the following types of information:
  - Number of passing and failing parts
  - Number of parts in each software bin and hardware bin
  - Average socket time
  - Average cycle time

Obtaining Information about the Test Execution

Complete the following tasks to use the Site Runtime Data, Batch Runtime Data, or Wafer Runtime Data objects to obtain run-time data for individual test sites for the current batch of parts on each test site or for the current wafer.

Note Some of the run-time data properties have valid values only at particular points during execution. Refer to the documentation of the property to determine when to access a property.

- In LabVIEW, use the Get Site Runtime Data VI to obtain the run-time data for an individual site. In .NET, call the `GetSiteRuntimeData` method on the Semiconductor Module Manager object and index into the array using the site number.
- Unbundle the Site Runtime Data cluster in LabVIEW or use the .NET properties and methods on the `SiteRuntimeData` object to obtain the following types of information about the site:
  - Start and end test times
  - Whether it is currently retesting or will retest the current part
  - Information about the current part
  - Wafer die coordinates
  - Hardware and software bin information the part will be assigned
- In LabVIEW, use the Get Batch Runtime Data VI to obtain run-time data for the current batch. In .NET, access the `BatchRuntimeData` property on the Semiconductor Module Manager object.

- Unbundle the Batch Runtime Data cluster in LabVIEW or use the .NET properties and methods on the `BatchRuntimeData` object to obtain the following types of information about the batch:
  - Start and end test times
  - Whether it is currently resting or will retest the current part
  - Whether this is the first or last batch in a wafer
  - The wafer run-time data
- Obtain the run-time data for the current wafer by calling the Get Wafer Runtime Data VI or by accessing the `WaferRuntimeData` property of the `BatchRuntimeData` .NET property.
- Use the values of the output terminals on the Get Wafer Runtime Data VI or use the .NET properties and methods of the `WaferRuntimeData` object to obtain the wafer ID and other information about the wafer.

See Also

Creating or Obtaining the Semiconductor Module Manager Object

Structure of the MainSequenceResult PropertyObject (TSM)

Use the MainSequenceResult parameter in the Model Plugin – UUT Done entry point in a custom model plug-in to obtain the test results for a part. The MainSequenceResult is a TestStand `PropertyObject` container that contains the test results for the `MainSequence` sequence in the test program. Use the TestStand core API to access individual test results.

The `MainSequenceResult` includes the following properties:

Name	Type	Description
Status	String	Status of the `MainSequence` sequence. Possible values are `Passed`, `Failed`, `Terminated`, and `Error`.
Error	Error (Container)	Contains error code and message if an error occurred.

<table>
<tr><td>TS.StartTime</td><td>Number</td><td>Time that execution of the sequence began, in seconds, since the TestStand Engine started. You can add this value to <code>Engine.SecondsAtStartIn1970UniversalCoordinatedTime</code> to convert to a universal time base.</td></tr>
<tr><td>TS.TotalTime</td><td>Number</td><td>Time in seconds to execute the <code>MainSequence</code> sequence.</td></tr>
<tr><td>TS.SequenceCall</td><td>Container</td><td>This property has the following subproperties:
<table>
<tr><th>Subproperty</th><th>Type</th><th>Description</th></tr>
<tr><td>SequenceFile</td><td>String</td><td>Absolute path of the test program sequence file.</td></tr>
<tr><td>SequenceFileVersion</td><td>String</td><td>Version of the test program sequence file.</td></tr>
<tr><td>Status</td><td>String</td><td>Do not use this property because it does not reflect errors if they occur.</td></tr>
<tr><td>ResultList</td><td>Array of Results</td><td>Individual step results the test program gener</td></tr>
</table>
</td></tr>
</table>

				ates. Each item is a `Result` container, which includes a set of properties common to all results and a set of properties specific to the type of step that produced the result.

The following table includes commonly used properties in `Result` containers.

Name	Type	Description
Error	Error (Container)	Contains error code and message if an error occurred.
Status	String	Status of the step. Possible values are `Passed`, `Failed`, `Done`, `Skipped`, `Terminated`, and `Error`. A status of `Skipped` indicates that the step did not execute.
TS.StartTime	Number	Time that the step started executing, in seconds, since the TestStand Engine started.
TS.TotalTime	Number	Time in seconds to execute the step.
TS.Index	Number	Index of the step in the step group of the sequence.
TS.StepName	Number	Name of the step.

TS.StepGroup	String	Name of the step group. Possible values are `Setup`, `Main`, and `Cleanup`.
TS.StepId	Number	Unique ID of the step.
TS.StepType	String	Name of the step type. Use this property to determine what type of step the result belongs to. If this string has the value `NI_SemiconductorModule_MultiTest`, the `Result` container includes an Evaluations property, as described below.
AdditionalResults	Array of Container	This property exists only if the step contains a list of additional results on the Additional Results Panel. Each item in the array corresponds to a row in the table in the panel.
SemiconductorCommon.Sites	String	This property exists only on results of Semiconductor Multi Test steps and Semiconductor Action steps. This string stores a comma-separated list of sites tested in the current thread, as shown in the Multisite Execution Diagram of the step.
SemiconductorCommon.RaisedAlarms	String	This property exists only on results of Semiconductor Multi Test steps and Semiconductor Action steps. This string stores a comma-separated list of alarms that were raised by the step.
Evaluations	Array of `NI_SemiconductorModule_Evaluation`	This property exists only on results of Semiconductor Multi Test steps. Each evaluation item contains the result of evaluating test limits on a test listed on the Tests tab of the Semiconductor Multi Test step. To reduce memory requirements, not all Semic

<table>
<tr><td></td><td></td><td>onductor Multi Test step properties exist in the result list.</td></tr>
<tr><td>TS.SequenceCall</td><td>Container</td><td>This property exists only on results of Sequence Call steps and contains the results from steps in the called sequence. This property has the following subproperties:
<table>
<tr><th>Subproperty</th><th>Type</th><th>Description</th></tr>
<tr><td>SequenceFile</td><td>String</td><td>Absolute path of the sequence file that contains the called sequence.</td></tr>
<tr><td>SequenceFileVersion</td><td>String</td><td>Version of the sequence file that contains the called sequence.</td></tr>
<tr><td>Sequence</td><td>String</td><td>Name of the called sequence.</td></tr>
<tr><td>Status</td><td>String</td><td>The status of the sequence call step.
Note If an error</td></tr>
</table>
</td></tr>
</table>

				occurs in the called sequence this property is not set to `Error`.

See Also

TestStand Standard Result Properties

TestStand Semiconductor Module Application .NET API

April 2022, 375355J-01

This help file contains detailed information about the TestStand Semiconductor Module™ (TSM) Application .NET API.

Note If you open help files directly from the `<TestStand>/Doc/Help` directory, NI recommends that you open `TSHelp.chm` first because this file is a collection of all the TestStand help files and provides a complete table of contents and index.

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window.

Glossary (TSM)

A B C D G H I L M P R S T V W

A

active site	Site that is currently testing DUTs and not disabled.

B

batch	A set of DUTs you test simultaneously.
bin definitions file	Defines the hardware bins and software bins, defines how the software bins relate to hardware bins, and defines the default software bins for the test program main sequence file.
binning	Process of assigning a software bin and hardware bin to a tested DUT.

C

channel	Connection to a data acquisition system or to an instrument.
channel group	A synchronized group of channels.
configuration	Defines values for conditions that a test program can reference at run time and the test limits file that loads before running a test lot. A test program can use multiple configurations to implement multiple test flows using the same sequences and code modules. For example, you can create configurations for Hot and Cold flows or for QA and Production lots.
connection	An entry in a pin map file that defines the relationship of a pin to a channel on an instrument.
custom instrument	An instrument that TSM does not natively support.
cycle time	The socket time plus the index time. You can use the TSM Application API to obtain cycle time

information.
See also Execution Timing Overview.

D

device — Item you are testing.

DIB — Device interface board.

DUT — Device under test.
See also part.

DUT pin — One of the following:

- A specific pin on the DUT.
- A resource on the tester or DIB that has instrument connections and that is associated with one or more sites. This resource can have one connection per site or can have one connection per group of sites.

See also system pin.

G

grading — Process in which the test program evaluates a DUT with different test criteria and assigns a pass bin to the DUT depending on the level of criteria the DUT met.

H

handler — Places DUTs on the test head for testing, removes the DUTs from the test head after testing completes, and places the DUTs in a hardware bin, depending on the test results.

handler/prober index time — Time spent by a handler to remove and bin the tested parts and place new parts to be tested or by a prober to move the probes from the current position to the position of the next dies to be tested. Index time is measured by the time elapsed between sending the end-of-test notification to the handler or prober and

	receiving the start-of-test (SOT) notification from the handler or prober. **See also** Execution Timing Overview.
hardware bin	Physical location into which handler places a DUT after testing.
hardware configuration file	Defines the configuration of the instruments in the test system.

I

index time	See handler/prober index time.
inline QA	Performing one or more quality assurance tests within a standard test sequence.
instrument	Equipment used to test devices. TSM supports NI and custom instruments.

L

lot	**See also** test lot.

M

multisite	Testing multiple DUTs at the same time in parallel to improve tester efficiency.

P

part	The item to test. **See also** DUT.
pin	Input or output of a connection to a device you are testing.
pin group	A collection of pins defined in a pin map that can be specified in a Semiconductor Multi Test step test or passed as a parameter to the TSM Code Module API.
pin map	Defines the instrumentation on the tester, defines the pins on the DUT, and defines how the DUT pins are connected to the tester instrumentation for each test site.

prober	Tests integrated circuits on a wafer.
PTE	Parallel test efficiency.

R

result	Data that is logged for analysis or used to evaluate the pass/fail status of a DUT.

S

Semiconductor Module context	Required input to all TSM Code Module API VIs and .NET methods that represents a subset of pins, sites, and instruments on a test system.
site	Physical location on a tester for testing DUTs. **See also** test socket. TSM site numbers do not always directly correspond to test socket indexes. Use the Get Site Runtime Data VI or the `GetSiteRuntimeData` .NET method of the TSM Application API or use the Get Test Information step to obtain specific site numbers.
site relay	A relay on the tester or DIB that is connected to a relay driver module and that is associated with one or more sites. A site relay can have one connection per site or can have one connection per group of sites. **See also** system relay.
socket time	Time spent by the DUT in the test socket, as measured by the time elapsed between receiving the start-of-test (SOT) notification from the handler or prober and sending the end-of-test (EOT) notification to the handler or prober. You can use the TSM Application API to obtain socket time information. **See also** cycle time and tester index time. **See also** Execution Timing Overview.
software bin	Virtual bin you define in software to store information about a DUT or testing results for a DUT before assigning the DUT to a hardware bin after testing. Use a bin definitions file to define

	the relationship between software bins and hardware bins.
software fail bin	Software bin associated with a hardware bin of the `Fail` type. The Software Bin column on the Tests tab of the Semiconductor Multi Test step lists only software fail bins.
software pass bin	Software bin associated with a hardware bin of the `Pass` type.
STDF	Standard Test Data Format. A standard file format for storing semiconductor test result data.
STS	NI Semiconductor Test System.
subsystem	A set of sites and system resources on the tester that can operate independently. A Semiconductor Multi Test step automatically calculates and creates subsystems depending on the active sites for the step and the pins required to complete the test.
system pin	A resource on the tester or DIB that is connected to an instrument. A system pin has a single connection and is associated with all sites. You can also use DUT pins for shared resources by specifying the sites that share the resource in the connection. Use a DUT pin instead of a system pin if you need to burst patterns to the pin using the NI-Digital Pattern instrument driver. **See also** DUT pin.
system relay	A relay on the tester or DIB that is connected to a relay driver module and that is associated with all sites. A system relay has a single connection for all sites. **See also** site relay.

T

test cell	The entire physical area for testing DUTs. The test cell includes the tester (or test station), the DUT handler or prober, the operator, and

	anything else physically located in the test area that has an effect on the test cell operation.
test code	A program module, such as a DLL or VI, that contains one or more functions that perform a specific test or other action.
test condition	Specifies historical information, descriptive information, such as DUT numbers or package types, and conditions under which to test the DUTs, such as temperature or voltage. The test program can use test conditions to determine how to execute tests. For example, test conditions might dictate which steps execute, what temperature to apply to a DUT, what voltage to use, and so on.
test limits file	Defines test limits the test program loads before running a test lot. The test program replaces test limits in test steps in the sequence file with those specified in the test limits file. You can embed test limits in the sequence file to prevent viewing or tampering with the limits.
test lot	Set of DUTs tested during a single testing session.
test number	Integer that uniquely identifies a specific test instance.
test program	The set of information that specifies how to execute the test. A semiconductor test program requires a main sequence file, optional subordinate sequence files, a pin map file, a bin definitions file, and code modules.
test program main sequence	Specifies the tests and test limits and determines which code modules to call to test a DUT. The main sequence file refers to a pin map file and bin definitions file that the test program uses during execution. The sequence file contains at least one `MainSequence` sequence and can optionally contain one or more subsequences. A subsequence can call its own code modules, but you can specify only one pin map file or bin definitions file for a

	test program. You can use multiple sequences in a test program to keep the test code modular and organized.
test socket	An execution thread in TestStand for testing a DUT for an associated site. **See also** Process Model Thread Types. TSM site numbers do not always directly correspond to test socket indexes. Use the Get Site Runtime Data VI or the `GetSiteRuntimeData` .NET method of the TSM Application API or use the Get Test Information step to obtain specific site numbers.
test station	A complete test implementation that production operators, test engineers, and system engineers use to perform tests.
test step	An instance of the Semiconductor Multi Test step type that performs one or more parametric or functional tests. A test step calls a code module implemented in LabVIEW or .NET to control the instrumentation on the tester, take measurements from the DUT, and pass measurement values back to the Semiconductor Multi Test step.
tester	The hardware solution for executing semiconductor tests. The tester is connected to a handler, which places DUTs in a test site or a prober that probes a wafer. The tester includes on-board instruments that perform measurements. The tester executes the tests the tester software defines.
tester index time	Time required to update the operator interface, process results for reports and data logs, and perform other tasks that the tester must complete before starting the next testing cycle. **See also** Execution Timing Overview.
tester software	The software solution installed on a tester that defines the test program and provides software tools for configuring and executing tests, such as a handler/prober driver for communicating with a handler or prober.

TSM	TestStand Semiconductor Module™

V

virtual pin	A DUT pin that does not physically exist but that you create in the pin map so you can map multiple instruments to the same physical DUT pin.

W

working site	The test socket that executes the code module. When testing multiple sites, the working site is assigned to the first site that reaches a step for the set of sites that must execute together during execution and, when you configure the test step to execute all sites in a single thread, is therefore the only site that executes a copy of the code module.