Measurement Plug-Ins

2025-03-13

Contents

Measurement Plug-In Overview

Measurement Plug-In helps you create reusable measurements for any supported device. Use Python or LabVIEW to generate a new measurement plug-in and define your measurement logic. Deploy your measurement plug-in to perform interactive validation in InstrumentStudio and automated testing in TestStand.

Measurement Plug-In Workflow

This documentation supports developing measurement plug-ins and using measurements in low-code applications.

- Develop reusable measurement plug-ins:
  - Develop measurement logic in your preferred software language. Measurement Plug-In services help to manage driver sessions, allowing you to create software measurements that emphasize intuitive interaction with a physical DUT.
  - Validate measurement behavior from your development environment or interactively in InstrumentStudio.
  - Use LabVIEW or the Measurement Plug-In UI Editor to create a graphical user interface that loads when you open your measurement in InstrumentStudio.
  - Deploy measurement plug-ins so that they are statically registered with the Measurement Plug-In discovery service and available in TestStand and InstrumentStudio.
- Use measurements interactively:
  - Use InstrumentStudio to run a measurement and perform hardware validation via a graphical UI.
  - Open a measurement in InstrumentStudio from TestStand to debug a measurement step in your automated sequence.
- Use measurements in automated testing:
  - Use TestStand to run measurements as part of an automated test sequence.
  - Register a pin map so that the Measurement Plug-In session management service can initialize and manage driver sessions for NI drivers.
  - Initialize and manage instrument driver sessions efficiently with multiple measurements in a single automated test sequence.

Related concepts:

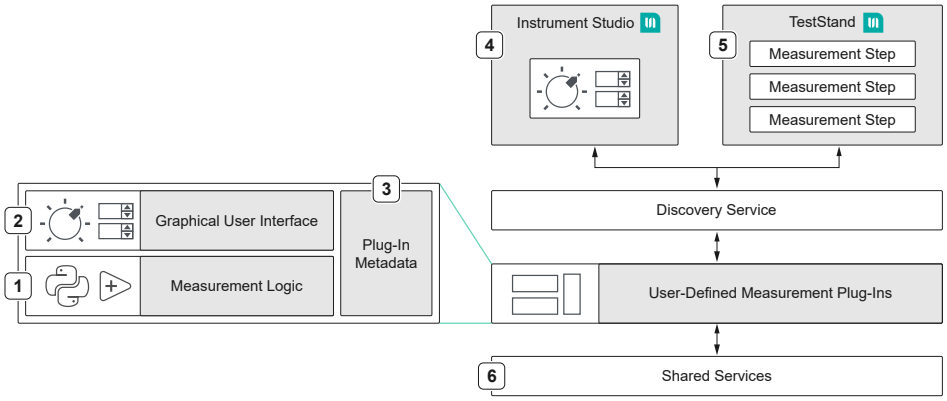
- Understanding Measurement Plug-In Behavior

Related reference:

- Measurement Plug-In Architecture

Measurement Plug-In Architecture

.



1. Use Python or LabVIEW to generate a new measurement plug-in and define your measurement logic.
2. Create a graphical UI for users who interact with your measurement in InstrumentStudio.
3. Deploy your measurement plug-in with a few simple steps. The Measurement Plug-In discovery service registers your measurement so that you can use it in supported applications.
4. Use InstrumentStudio to open and run measurements interactively for debugging or validation.
5. Use TestStand to run measurements automatically as part of a test sequence.
6. Measurement Plug-In microservices provide:
   - Uniform behavior for all measurements, regardless of source language.
   - The ability to monitor measurements when they run from a separate application.
   - The ability to transfer measurement configuration values between supported applications.
   - Support for pin maps, which allow you to associate device I/O with user-

defined pin names and site names.
    - The ability to configure and initialize instrument driver sessions based on information stored in a pin map.

Related tasks:

- Using Measurements in Custom Applications

Understanding Measurement Plug-In Behavior

A measurement plug-in consists of measurement logic, metadata, and an optional UI. A measurement plug-in runs as a service. The measurement logic does not execute when your measurement service starts.

A measurement plug-in can be deployed and statically registered with the Measurement Plug-In discovery service. The Measurement Plug-In discovery service makes registered measurement plug-ins visible in supported applications.

To execute your measurement logic, do one of the following:

- Run your measurement logic from your measurement development environment.
- Run your measurement interactively by clicking **Run** on the measurement UI toolbar in InstrumentStudio or the Measurement Plug-In UI Editor.
- Run an automated test sequence that includes your measurement as a step using TestStand.

Note While a measurement service runs, there are multiple ways to execute the associated measurement logic. The measurement UI toolbar indicates where active measurement data originates.

When you execute your measurement logic, the measurement service is called and your configuration values are passed to the measurement service. When the measurement logic completes execution, the measurement service returns measurement data.

Pin Maps and Hardware-Based Measurements

Pin maps associate physical device I/O with user-defined pin names and site names. For Measurement Plug-In measurement plug-ins, a pin map allows you to address device I/O while Measurement Plug-In services use information from the pin map to simplify driver session management. Pin maps enhance the flexibility of your measurement and allow the measurement user to focus on the DUT in a more intuitive way.

Use InstrumentStudio to create a pin map for your application. For more information about how pin maps are used, refer to ***Understanding the Driver Session Management Service*** and ***Understanding the Pin Map Service***.

Note A pin map is not required to take measurements.

Related tasks:

- Running a Measurement from InstrumentStudio
- Running a Measurement from TestStand
- Using Measurements in Custom Applications

Related information:

- Statically Register a Measurement Service

New Features and Changes

Learn about updates—including new features and behavior changes. Product features on github may update more frequently than indicated in these notes.

2024 Q4

New Features

- Added support for measurement plug-in versioning.
- Added support for displaying a link to measurement plug-in documentation in InstrumentStudio and TestStand.
- Added support for plug-in client creation.

Related concepts:

- Plug-In Versioning

Related reference:

- Developing a Measurement Plug-In Client with LabVIEW
- Developing a Measurement Plug-In Client with Python

2024 Q3

Changes

- MeasurementLink technology will now be included as measurement plug-ins for use with other NI products.

New Features

Pin maps are no longer required to take measurements.

InstrumentStudio

Added support for using measurement plug-ins in InstrumentStudio Professional. See ***Measurement Plug-Ins*** in the InstrumentStudio manual for more information.

MeasurementLink 2024 Q2

MeasurementLink 2024 Q2 includes the following new features and changes:

- Return multiplexer (MUX) information from the session manager reserve function (python).
- Add support for pin and relay groups to the session manager service and session manager APIs (LabVIEW and python).

MeasurementLink 2024 Q1

MeasurementLink 2024 Q1 includes the following new features and changes:

- Additional LabVIEW Vis and Python methods to simplify session management.
- New Update Pin Map custom step in TestStand. The Update Pin Map step simplifies registering a pin map in TestStand.
- Support for XYGraph output data type for MeasurementLink UIs.
- VISA gRPC and session support.

MeasurementLink 2023 Q4

MeasurementLink 2023 Q4 includes the following new features and changes:

- Support for TestStand 2023.
- Support for optional annotations in the Service Configuration File for description, collection and tags. These are used for better measurement organization in InstrumentStudio, TestStand and the MeasurementLink UI Editor.
- Support for ring controls in the measurement UI.
- Allow running measurements continuously in InstrumentStudio (as a MeasurementLink preview feature).

MeasurementLink 2023 Q3

MeasurementLink 2023 Q3 includes the following new features and changes:

- Support for gRPC connections in the NI-DAQmx Python API.
- Support for updating the interactive measurement UI progressively while the measurement runs. This update does not change measurement step behavior in TestStand.
- Support for `enum` data type and controls in measurements.

MeasurementLink 2023 Q2

MeasurementLink 2023 Q2 includes the following new features and changes:

- Support for native gRPC interfaces in LabVIEW for the following drivers:
  - NI-Digital Pattern Driver
  - NI-DMM
  - NI-FGEN
  - NI-SCOPE

MeasurementLink 2023 Q1

Initial release.

The following table indicates the NI driver software (and available interfaces) supported in the first MeasurementLink release. An updated list of supported driver software can be found elsewhere in this manual.

Supported NI driver software	Support for native gRPC interface (Python)	Support for native gRPC interface (LabVIEW)
NI-DAQmx	–	–
NI-DCPower	✔	✔
NI-Digital Pattern Driver	✔	–
NI-DMM	✔	–
NI-FGEN	✔	–

Supported NI driver software	Support for native gRPC interface (Python)	Support for native gRPC interface (LabVIEW)
NI-SCOPE	✔	–
NI-SWITCH	✔	–
NI-VISA (GPIB, serial, LXI interfaces)	–	–

The following list reflects the data types supported in the first MeasurementLink release. An updated list of supported data types can be found elsewhere in this manual.

- Signed integers
  - Int32
  - Int64
- Unsigned integers
  - UInt32
  - UInt64
- Floating-point numbers
  - Float (32-bit floating-point number)
  - Double (64-bit floating-point number)
- Boolean
- String
  - Pin (string type specialization)
  - Path (string type specialization)
- 1D array (supported for all base datatypes and type specializations)

Installing Measurement Plug-In

To install Measurement Plug-In:

1. Install InstrumentStudio. The InstrumentStudio version must match your Measurement Plug-In version.
2. Install TestStand 2021 or later if you will use measurements in automated test applications. Measurement Plug-In is only compatible with 64-bit versions of TestStand.
3. Install Measurement Plug-In.

   > **Note** The Measurement Plug-In UI Editor is an optional feature that developers can use to create a measurement UI without using LabVIEW.

4. To develop measurement plug-ins, install dependencies for your software language:
   - Install the Python Measurement Development Dependencies
   - Install the LabVIEW Measurement Development Dependencies

Examples

Measurement Plug-In provides example measurements for each supported software language.

Examples are available as an asset within the Measurement Plug-In releases. To download and install examples, use the following links along with the documentation located in the associated repository.

- Python examples: https://github.com/ni/measurement-plugin-python
- LabVIEW examples: https://github.com/ni/measurement-plugin-labview

Python Measurement Development Dependencies

You must install this software to develop Measurement Plug-In measurements using Python.

1. Ensure that Python 3.8 or later is installed.
2. Install the measurement service package:

   ```
   python -m pip install  ni_measurement_plugin_sdk
   ```

   This will also install all remaining Python dependencies.

LabVIEW Measurement Development Dependencies

You must install LabVIEW add-on packages to develop Measurement Plug-In measurements using LabVIEW. The required packages are available from vipm.io.

Complete the following steps to install LabVIEW add-on packages using vipm.io.

1. Navigate to the Measurement Plug-In package list on the vipm.io website. You can also search for `Measurement Plug-In` to see a list of available packages.
2. On the vipm.io website, click the Measurement Plug-In add-on. The add-on page opens.

   > **Note** Installing the Measurement Plug-In Generator add-on will also install the Measurement Plug-In Service add-on as a dependency.

3. Click **Install with VIPM**. If prompted, allow vipm.io to open the link with the associated app. VIPM opens and displays the Measurement Plug-In Generator add-on.
4. In the VIPM application, click **Install**. VIPM displays a list of pending actions.

   > **Note** By default, VIPM installs dependencies for the selected add-on.

5. Click **Continue**.
6. If you install dependencies, VIPM will prompt you to agree to download additional software. Click **I Agree** to continue.
7. LabVIEW launches and VIPM displays add-on installation activity. When installation is complete, click **Finish** in VIPM.

Add-on installation is complete. Close VIPM and use LabVIEW to develop your measurement.

Related information:

- Accessing LabVIEW Add-ons with the VI Package Manager (VIPM) Software (Windows)
- Measurement Plug-Ins Package List on vipm.io

Developing a Measurement Plug-In with Python

This topic outlines the required steps for developing a measurement plug-in in Python. Use the Python examples in github and the remaining topics in this manual to understand how to develop a working measurement for your application.

Note If you are developing a measurement plug-in for use with TestStand, review the ***Running a Measurement from TestStand*** topic to understand additional design considerations.

Related reference:

- Python Measurement Development Dependencies

Related information:

- Measurement Plug-In SDK Examples for Python on github

Creating a Python Measurement Plug-In

Complete the following steps to generate a new measurement plug-in with Python. Ensure that you have installed the Python development dependencies before you begin.

1. Open a command prompt.
2. Run `ni_measurement_plugin_generator measurement name --directory-out path` to create a new measurement service.

Note If you omit the `--directory-out` parameter the new measurement plug-in folder appears in the current directory.

Note You can specify additional parameters. View the ***Python***

> ***Measurement Plug-In Generator Parameters*** topic for details.

A folder containing the new measurement plug-in appears in the specified directory.

Configuring a Python Measurement Plug-In

You must modify a generated Python measurement plug-in to meet your specific needs.

1. Navigate to your measurement folder and open `measurement.py`. This file defines your measurement logic.
2. Add drivers and packages to the import list as necessary. You can create a measurement for any hardware with an accessible I/O library.
3. Optionally, specify `version` and `ui_file_path` values for your measurement.
4. Edit the measurement configuration (input parameters).
   a. Use the `configuration()` decorator to define a configuration. Configuration decorators must be listed in the same order as the parameters in the measurement function signature.

      Configuration decorator syntax:

      ```
      @meas_name_measurement_service.configuration("DisplayName",
      nims.DataType.Type, "DefaultValue")
      ```

   b. Use the `output()` decorator to define an output. Output decorators must be listed in the same order as the measurement function return (or yield) values.

      Output decorator syntax:

      ```
      @meas_name_measurement_service.output("DisplayName", nims.DataType.Type)
      ```

5. To update the interactive UI while the measurement runs, use the yield keyword. The `ui_progress_updates` example demonstrates this feature.
6. Implement your measurement logic within the measurement function. If you are developing a measurement plug-in for use with TestStand, review the ***Running a Measurement from TestStand*** topic to understand additional design considerations.

7. Save and close `measurement.py`.
8. Optionally, edit the `*.serviceconfig` file to customize the display name, service class, or provided interface for your measurement.

Setting Environmental Variables

You can set environment variables to configure `ni_measurement_plugin_sdk` settings. You can create a `.env` file to set environmental variables. A `.env` file is a text file containing environment variables in the form `VAR=value`. The `.env` may be located in these locations, listed in priority order:

- The measurement service's current working directory or one of its parent directories. For example, you can place a `.env` file in `<ProgramData>\National Instruments\Plug-Ins` to configure statically registered services.
- The path value set in the `.serviceconfig` file.
- The path of the Python module calling into `ni_measurement_plugin_sdk`. This behavior provides support for TestStand code modules.

For example, the modular instrument `initialize_${driver}_sessions(s)` methods allow you to use `.env` settings to override the IVI option string and specify simulation options.

```
# Add this to your .env file to enable NI-DCPower simulation with PXIe-4141
instruments.
MEASUREMENT_PLUGIN_NIDCPOWER_SIMULATE=1
MEASUREMENT_PLUGIN_NIDCPOWER_BOARD_TYPE=PXIe
MEASUREMENT_PLUGIN_NIDCPOWER_MODEL=4141
```

For a complete reference of configurable settings, refer to the `.env.sample` file located in the root folder of the latest Measurement Plug-In release examples asset.

Use the Measurement Plug-In UI Editor to create a UI for your measurement.

Starting a Python Measurement Service

To manually run your Python measurement plug-in as a service, open a command prompt and run `start.bat` from your measurement folder.

Note Starting your measurement service from a development environment temporarily displaces any statically registered measurement service with the same service class. When you stop the service, the statically registered instance is restored upon being called. This behavior is useful for rapid debugging. Ensure a unique service class for each measurement service to avoid unexpected behavior.

To automatically run your measurement plug-in as a service, use the generated service configuration file to statically register your plug-in. You should statically register your measurement service when you deploy it to a production environment.

Statically Register a Measurement Service

The Measurement Plug-In discovery service automatically registers measurement services that are deployed to the Measurement Plug-In services folder. The discovery service continuously monitors this folder for changes.

You must do the following before you deploy your measurement plug-in:

- Create a batch file that invokes the measurement logic or compile the measurement project as an executable.
- Ensure that your measurement plug-in has a valid service configuration (`*.serviceconfig`) file.
  - LabVIEW measurement plug-ins can generate this file by running the included build specification.
  - For other scenarios, use the service configuration file template to create this file for your plug-in.

Complete the following steps to deploy your measurement plug-in and statically register your measurement service.

1. Ensure the `path` value in the service configuration file is correct for your batch file or executable.
2. Copy the measurement plug-in folder to the following location:
   `<ProgramData>\National Instruments\Plug-Ins\Services\`

Note The discovery directory is continously monitored. Service configuration file changes take immediate effect.

Once your measurement service is statically registered, the Measurement Plug-In discovery service makes it visible in supported NI applications.

Considerations when Deploying Python Measurement Plug-Ins

Be aware of the following considerations when deploying Python-based measurement plug-ins:

- For errors indicating a file cannot be found or accessed, do one of the following:
  - ensure that you have enabled Win32 long paths.
  - move the plugin or restructure the measurement plug-in to use shorter paths. Note that only the `*.serviceconfig` file must be deployed to the discovery folder. The file can reference paths outside the discovery folder.
- If you are using a virtual environment, recreate it in the deployed location. Do not move a virtual environment because some files may reference the path for the virtual environment.

Note To exclude virtual environment files from being adding to a plug-in library, create a file titled `.sericeignore` within the directory containing the Python plugin. Add the following line to this file:

```
.venv/*/**
```

This will exclude all virtual environment files from being published to the plug-in library. For more information, refer to ***Using a Plug-In Library***.

Service Configuration File Template

Use this template to create a service configuration file for your measurement plug-in.

A typical service configuration filename is `measurement plug-in name.serviceconfig`. The file contents are as follows:

```
{
```

```
  "services": [
    {
      "displayName": "display_name",
      "serviceClass": "service_class",
      "descriptionUrl": "description_url",
      "providedInterfaces": [
        "ni. ni_measurement_plugin_sdk .measurement.v1.MeasurementService",
        "ni. ni_measurement_plugin_sdk .measurement.v2.MeasurementService"
      ],
      "path": "service_filepath",
 "annotations": {
        "ni/service.description": "service_description",
        "ni/service.collection": "collection_name",
        "ni/service.tags": []
      }
    }
  ]
}
```

Object Name	Description
displayName	Specifies the display name of the measurement.
serviceClass	Class name for the measurement service. Also serves as the ID for the measurement service and must be unique across all measurement services.
descriptionUrl	Specifies a URL that contains more information about the measurement service.
providedInterfaces	Lists the measurement service interfaces. Do not edit this value.
path	Specifies the path to the measurement executable or start batch file.
ni/service.description	A short description of the measurement.
ni/service.collection	The collection that this measurement belongs to. Collection names are specified using a period-delimited namespace hierarchy and are case-insensitive.
ni/service.tags	Tags describing the measurement. This option may be repeated to specify multiple tags. Tags

Object Name	Description
	are case-insensitive.
`installPath`	Path to an executable or batch file used to install Python dependencies. For example, you might point to an `install.bat` file with the contents `poetry install --only main`.

Note

For measurement plug-ins sourced in LabVIEW, service configuration file values should match the values specified in your measurement plug-in project.

Python Measurement Plug-In Generator Parameters

The ni_measurement_plugin_generator tool requires the measurement plug-in name as the first parameter. This first parameter is unnamed.

The ni_measurement_plugin_generator tool accepts the following named parameters in any order using `--name value` syntax.

Parameter Name	Default Value	Description
`--d` or `--description-url`	“” (empty string)	Specifies a URL that contains more information about the measurement plug-in.
`--o` or `--directory-out`		Specifies the directory where your measurement plug-in folder is created. The generator creates a new folder with the name you specified in the Measurement Name parameter.
`--measurement-`	1.0.0.0	Specifies a version number in the form x.y.z.q.

Parameter Name	Default Value	Description
`version`		
`--s` or `--service-class`	“measurement name_Python”	Service Class that the measurement plug-in belongs to. Also serves to identify the measurement plug-in while it is running as a service.
`--u` or `--ui-file`	measurement name.measui	Specifies the file containing the UI for the measurement plug-in.

Python Measurement Plug-In File Descriptions

A new Python measurement plug-in consists of the following files:

File Name	Description
`*.serviceconfig`	Contains information required to run the measurement as a service.
`*.measui`	A ni_measurement_plugin_generator UI Editor file. Contains the measurement UI. If desired, you may replace the UI with a LabVIEW UI.
`*.measproj`	The measurement plug-in project file. Select this file to open the measurement in the ni_measurement_plugin_generator UI editor.
`measurement.py`	Contains the measurement logic and metadata. Edit this file to modify measurement behavior or to extend the measurement features.
`start.bat`	Calls Python and runs `measurement.py`.

Developing a Measurement Plug-In Client with Python

This section outlines the required steps for developing a measurement plug-in client in Python. Use the Python examples in github and the remaining topics in this manual to understand how to develop a working measurement plug-in client for your application.

Creating a Python Measurement Plug-In Client using Batch Mode

The client can be created using a one-line command by configuring the necessary options in Batch Mode.

1. Open a command prompt.
2. Enter the following command, specifying the name of the measurement service class: `ni_measurement_plugin_client_generator --measurement-service-class measurement_service_class_name`
3. **Optional:** You can specify additional parameters when running this command. For more details, refer to ***Python Measurement Plug-In Client Generator Batch Mode Parameters***. If no further parameters are required, run the command.
   A Python file containing the client class will be generated for the specified measurement service class. For details on using Python plug-in clients, refer to ***Python Measurement Plug-In Client Usage***

Python Measurement Plug-In Client Generator Batch Mode Parameters

The ni_measurement_plugin_client_generator tool requires the measurement plug-in service class as the first parameter. This parameter can be specified multiple times to create multiple plug-in clients.

The tool accepts the following named parameters in any order using the --name value syntax:

Parameter Name	Default Value	Description
-s, --measurement-service-class	-	Creates a Python Measurement Plug-In Client for the given measurement service. **Note** This parameter should not be used with the `--all` parameter.
-a, --all	False	Creates a Python Measurement Plug-In Client for all the registered measurement services.
-m, --module-name	Constructed from service class	Name for the Python Measurement Plug-In Client module. This value is ignored if --all is chosen.
-c, --class	Constructed from service class	Name for the Python Measurement Plug-In Client class in the generated module. This value is ignored if --all is chosen.
-o, --directory-out	`current_directory/module_name.py`	Output directory for Measurement Plug-In Client files.

Creating a Python Measurement Plug-In Client using Interactive Mode

Measurement plug-in clients can be created individually by providing the necessary inputs via the command line in interactive mode.

1. Open a command prompt.

2. Run the following command to list the registered and active measurements on the machine: `ni_measurement_plugin_client_generator --interactive`
   This command displays a numbered list of all registered measurement services.

```
1. Measurement A
2. Measurement B
3. Measurement C
```

3. Enter the number corresponding to the measurement you want to create a client module for. To create a client module for Measurement B from the example above, you would enter `2`.
4. Enter the name of the measurement client module and class, ensuring that the name follows Python's module naming conventions.
   A Python file containing the client class will be generated for the specified measurement service class. For more details, refer to ***Python Measurement Plug-In Client Usage***.

Python Measurement Plug-In Client Usage

The generated Python file will contain a class and public methods for the specified measurement. Before you can interact with a measurement via the plug-in client, the Measurement Service must be running. Refer to the following section for descriptions of the public methods you can use to interact with a measurement service:

- `register_pin_map`— Registers the pin map with the pin map service.
- `measure`— Interacts with non-streaming data measurements by accepting the actual input configured in the measurement and returning the measurement output.
- `stream_measure`— Same as `measure`, but used specifically for interacting with streaming data measurements.
- `cancel`— Aborts any ongoing `measure` or `stream_measure` call.

Note This API can only be used from a separate thread.

Developing a Measurement Plug-In with LabVIEW

This topic outlines the required steps for developing a measurement plug-in in LabVIEW. Use the LabVIEW examples in github and the remaining topics in this manual to understand how to develop a working measurement for your application.

> **Note** If you are developing a measurement plug-in for use with TestStand, review the ***Running a Measurement from TestStand*** topic to understand additional design considerations.

Related reference:

- LabVIEW Measurement Development Dependencies

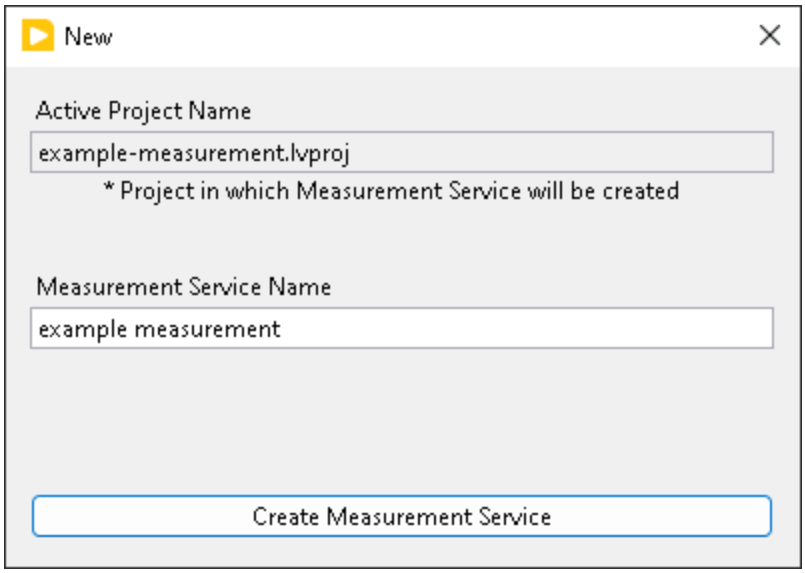
Related information:

- Measurement Plug-In SDK Examples for LabVIEW on github

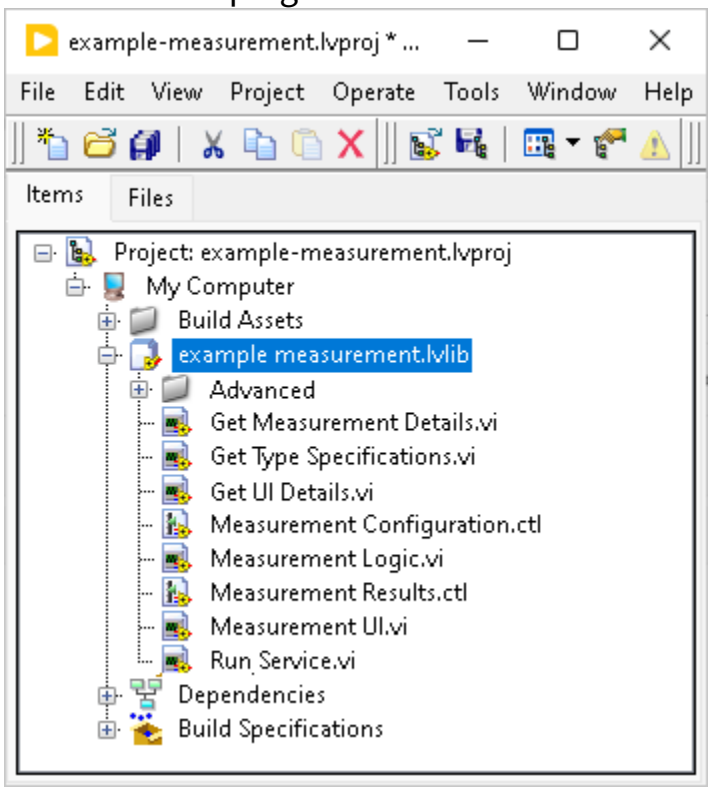
Creating a LabVIEW Measurement Plug-In

Complete the following steps to add a measurement plug-in to a LabVIEW project. Ensure that you have installed the LabVIEW development dependencies before you begin.

1. Open or create a LabVIEW project. Save the project.
2. In the Project Explorer window select **Tools » » Create Measurement Service**. A dialog box appears.
3. Enter a name for the new measurement plug-in.



4. Click **Create Measurement Service**.
   A new library is added to the LabVIEW project. This library contains your measurement plug-in.



Note The measurement plug-in library also includes a build specification to simplify the process of creating an executable, which you

can use to deploy your measurement plug-in.

Configuring a LabVIEW Measurement Plug-In

You must modify a generated LabVIEW measurement plug-in to meet your specific needs.

Note Necessary steps are also documented as bookmarked to-do sections. In the LabVIEW project window select **View** » **Bookmark Manager** and browse the **#MeasurementToDo** bookmarks. Double-click a bookmark to open the relevant VI and code location.

These steps assume that your LabVIEW project is already open.

1. Edit the measurement inputs.
   a. Within your measurement plug-in library, open the `Measurement Configuration.ctl` typedef file.
   b. Expand the **Measurement Configuration** cluster and add controls to represent input values. Change the value of a control to specify a default value for the measurement.
   c. Save and close the typedef file.
2. Edit the measurement outputs.
   a. Within your measurement plug-in library, open the `Measurement Results.ctl` typedef file.
   b. Expand the **Measurement Results** cluster and add controls to represent output values.
   c. Save and close the typedef file.
3. Update the measurement logic.
   a. Within your measurement plug-in library, open `Measurement Logic.vi`.
   b. Implement your measurement logic using the existing configuration, results, and error clusters.
      Refer to the Measurement Development Best Practices topic for an abstract overview of useful development patterns.
      To update the interactive UI while the measurement runs, use the Update Results.vi. The `UIProgressUpdates` example demonstrates this feature.
   c. Save and close the VI.

4. If you are developing a measurement plug-in for use with TestStand, review the ***Running a Measurement from TestStand*** topic to understand additional design considerations.
5. Update the measurement UI.
   a. Within your measurement plug-in library, open `Measurement UI.vi`.
   b. Use LabVIEW controls and indicators to implement your UI. To function, control and indicator names and datatypes must correspond to the inputs and outputs of your measurement logic. Note that you can use captions (which have no matching constraint) to supplement labels.
   c. Save and close the VI.
6. Optionally, update the measurement details.
   a. Within your measurement plug-in library, open `Get Measurement Details.vi`.
   b. Open and edit the block diagram. View the LabVIEW Measurement Details topic for a description of each field.

Setting Pin or Path Behavior

Use `Get Type Specializations.vi` to specify whether a string behaves as a pin or path. Setting this behavior is useful when you intend to interact with the measurement logic through a graphical UI.

1. Open `Get Type Specializations.vi`.
2. Open the block diagram.
3. Modify the existing Type Specializations cluster to add your parameter.
   a. Specify a Parameter Name that matches the name of the parameter in your measurement logic.
   b. Select the Type Specialization for your parameter.
4. If you selected `Pin` as the Type Specialization for your parameter, use the Annotations cluster to specify the instrument type.
   a. Specify a Key of `pin.instrument_type`.
   b. Specify a Value. Valid values for NI devices are listed in the comment on the block diagram.
5. Close the VI and save your project.

Starting a LabVIEW Measurement Service

To immediately run a LabVIEW measurement plug-in from the LabVIEW project, open and run the **Run Service** VI located in your measurement plug-in library. The VI runs your plug-in as a service and reports the measurement service port.

Note Starting your measurement service from a development environment temporarily displaces any statically registered measurement service with the same service class. When you stop the service, the statically registered instance is restored upon being called. This behavior is useful for rapid debugging. Ensure a unique service class for each measurement service to avoid unexpected behavior.

To automatically run your measurement plug-in as a service, create a LabVIEW executable and use it to statically register the measurement plug-in. You should statically register your measurement service when you deploy it to a production environment.

Creating an Executable for a LabVIEW Measurement

Use the build specification in your measurement library to package a LabVIEW measurement as an executable, which can then be configured to run as a service.

Complete the following steps to run the build specification for your measurement library:

1. In your LabVIEW project, expand **Build Specifications**.
2. To review or change the destination directory, right-click the build specification and select **Properties**. The destination directory is specified on the Information page.
3. Right click the nested build specification file and select **Build**.

An executable and its dependencies are built to the destination directory specified in the build specification.

Statically Register a Measurement Service

The Measurement Plug-In discovery service automatically registers measurement services that are deployed to the Measurement Plug-In services folder. The discovery service continuously monitors this folder for changes.

You must do the following before you deploy your measurement plug-in:

- Create a batch file that invokes the measurement logic or compile the measurement project as an executable.
- Ensure that your measurement plug-in has a valid service configuration (`*.serviceconfig`) file.
  - LabVIEW measurement plug-ins can generate this file by running the included build specification.
  - For other scenarios, use the service configuration file template to create this file for your plug-in.

Complete the following steps to deploy your measurement plug-in and statically register your measurement service.

1. Ensure the `path` value in the service configuration file is correct for your batch file or executable.
2. Copy the measurement plug-in folder to the following location:
   `<ProgramData>\National Instruments\Plug-Ins\Services\`

> **Note** The discovery directory is continously monitored. Service configuration file changes take immediate effect.

Once your measurement service is statically registered, the Measurement Plug-In discovery service makes it visible in supported NI applications.

LabVIEW Measurement Plug-In Project File Descriptions

Each LabVIEW measurement plug-in is created as a separate library (`lvlib`) in your project. The following table describes important files within that library.

File Name	Description
`Get Measurement Details.vi`	Specifies metadata for the LabVIEW measurement service.
`Get Type Specializations.vi`	Specifies information about the parameters in your measurement configuration. For example, you can use this VI to annotate a string control as a pin name specialization. The Parameter Name associates the annotation with the underlying parameter. Do not edit the front panel. Switch to the block diagram and make changes there.
`Get UI Details.vi`	Specifies information about the measurement UI file, such as the UI file name.
`Measurement Configuration.ctl`	Defines measurement input parameters with LabVIEW controls. The default control is a double array labeled **Array in**.
`Measurement Logic.vi`	Defines measurement logic. This template file is already configured to use `Measurement Configuration.ctl` and `Measurement Results.ctl` as input and output.
`Measurement Results.ctl`	Defines measurement return values with LabVIEW indicators. The default indicator is a double array labeled **Array out**.
`Measurement UI.vi`	Defines the measurement user interface. To function, control and indicator labels and datatypes must correspond with `Measurement Configuration.ctl` and `Measurement Result.ctl`.
`Run Service.vi`	Initializes your measurement as a service and returns a listening port.

LabVIEW Measurement Plug-In Details

This topic provides additional information about the controls found in `Get Measurement Details.vi`.

Detail Name	Default Value	Description
Display Name	" <measurement name>_LabVIEW"	Specifies the display name of the measurement.
Version	1.0.0.0	Specifies a version number in the form x.y.z.q.
Service Class	"<measurement name>_LabVIEW"	Class name for the measurement service. Also serves as the ID for the measurement service and must be unique across all measurement

Detail Name	Default Value	Description
		services.

Developing a Measurement Plug-In Client with LabVIEW

This section outlines the required steps for developing a measurement plug-in client in LabVIEW. Use the LabVIEW examples in github and the remaining topics in this manual to understand how to develop a working measurement plug-in client for your application.

Creating a LabVIEW Measurement Plug-In Client

Complete the following steps to create a measurement plug-in client using LabVIEW.

Make sure LabVIEW development dependencies are installed before you begin.

1. Open or create a LabVIEW project and save the project file.
2. In the Project Explorer window, go to **Tools** » **Plug-In SDKs** » **Measurements** » **Create Measurement Plug-In Client**.
   A dialog box title Create Measurement Plug-In Client displaying all registered services opens.
3. Select one more more Measurement services from the list.
4. Click **Create Measurement Plug-In Client(s)**.
   A new library is added to the LabVIEW project containing your measurement plug-in client.

Refer to ***LabVIEW Measurement Plug-In Client Project File Descriptions*** for more details about the generated measurement plug-in client library.

LabVIEW Measurement Plug-In Client Project File Descriptions

Each LabVIEW measurement plug-in client is created as a separate library (`.lvlib`) in your project. The following sections describe important files within that library.

- `Measurement Configuration.ctl`— Defines measurement input

parameters with LabVIEW controls. This control is a copy of the Measurement Configuration.ctl in the Measurement Plug-In.
- `Measurement Results.ctl`— Defines measurement return values with LabVIEW indicators. This control is a copy of the Measurement Results.ctl in the Measurement Plug-In.
- `Create Measurement.vi`— Creates a client to communicate with the Measurement Plug-In Service and registers the measurement metadata for the created client.
- `Measure.vi`— Invokes the measurement logic and returns the result.
- `Close Measurement.vi`— Closes the client connection with the Measurement Plug-In.
- `Run Client.vi`— A wrapper VI that demonstrates the usage of `Create Measurement.vi`, `Measure.vi` and `Close Measurement.vi` present in the client library. It communicates with the Measurement Plug-In Service and returns the measurement result values.

Registering a Pin Map for a LabVIEW Measurement Plug-In Client

If your measurement service requires a pin map, you must register that pin map before performing a measurement. To register a pin map file for a pin-based measurement plug-in, complete the following steps.

1. Create a new VI.
2. From the Measurement I/O palette, select and place the `Register Pin Map.vi` in the new VI.
3. Provide the pin map path as input to the `Register Pin Map VI`.
4. Wire the `pin map id` output of the `Register Pin Map VI` to the input of the `Run Client VI`.
5. Configure your measurement and run the new VI.

Running a LabVIEW Measurement Plug-In Client

Complete the following step to run a LabVIEW measurement plug-in client from your LabVIEW project.

Open and run the `Run Client.vi` located in your measurement plug-in client library. This VI communicates with the Measurement Plug-In Service and returns the measurement results.

Note Ensure that the Measurement Service is running when interacting with measurements via the Measurement Plug-In Client.

Developing a User Interface for a Measurement Plug-In

Use the UI Editor to quickly create a user interface for your measurement plug-in. UI element datatypes must correspond with the datatypes and labels specified within your measurement logic.

Note You can also create a UI with LabVIEW and associate it with any measurement. To function, control and indicator labels and datatypes must correspond to the inputs and outputs of your measurement logic. Note that you can use captions to display text that is different from the label.

Creating a UI File

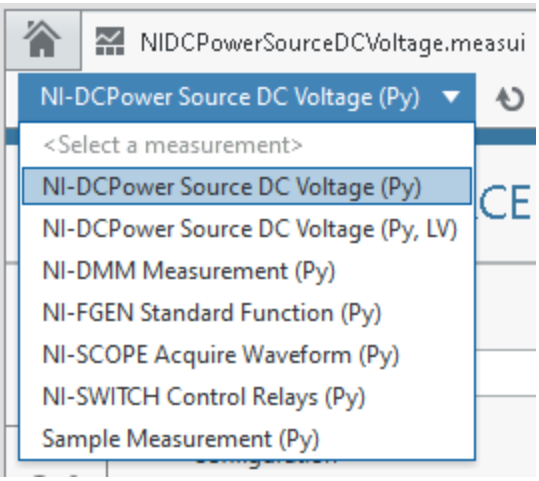
Complete the following steps to create a UI:

1. Launch UI Editor. Search installed applications or navigate to **National Instruments** » **Measurement Plug-In UI Editor** in the Windows start menu.
2. Select **File** » **New** » **Measurement UI**. A new project is created and a `*.measui` file opens for editing.

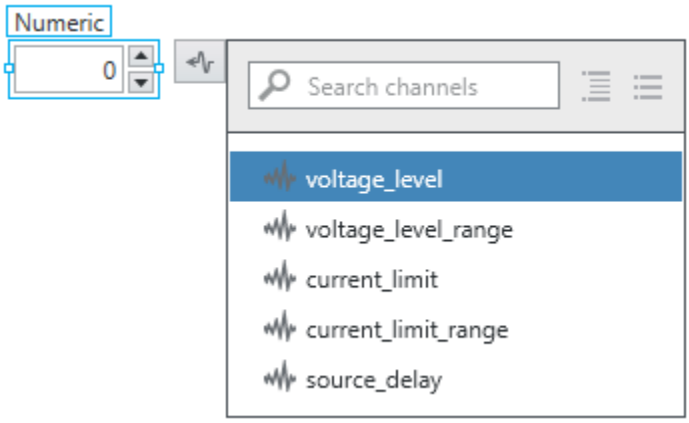
Note To rename project files, right-click the file in the Project Files pane and select **Rename**.

Linking the UI with Measurement Data

1. In UI Editor, select your measurement from the drop-down list below the file tab bar. If your measurement is not listed, ensure that your measurement service is running.



2. Use the left rail of the UI pane to add controls and indicators to the UI.
3. Use the **Data source** selector to associate measurement data with a UI element. The selector automatically appears when you add an element to the UI and can be accessed by hovering over a UI element.



Note Selecting a data source automatically updates the name of the UI element. You can manually change the name without disrupting the data source association.

4. Save your completed UI and edit your measurement to associate it with the new UI.

Note It is possible to associate a `*.measui` UI with a LabVIEW measurement in place of the default `Measurement UI.vi`. In your measurement service library, edit `Get UI Details.vi` to specify the path to your UI.

Specify the Measurement UI

You can associate a LabVIEW VI or a Measurement UI Editor interface (`*.measui`) with your measurement service.

Complete the following steps to associate a UI with a Python measurement service.

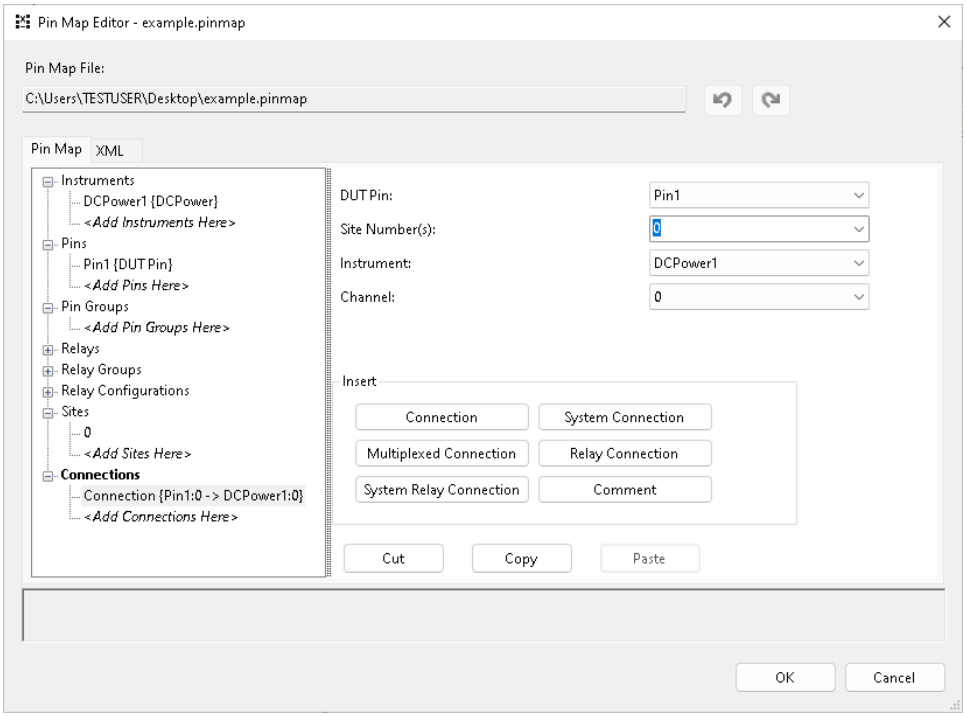
1. Open `measurement.py` for editing.
2. Locate the `measurement_info` variable declaration.
3. Edit the `ui_file_path` value with the path to your `*.measui` or `*.vi` file.

To specify the UI for a LabVIEW measurement service, update `Measurement UI.vi` in the measurement service library within your LabVIEW project. To associate a Measurement UI Editor interface (`*.measui`) with your LabVIEW measurement service, edit `Get UI Details.vi` in your measurement service library within your LabVIEW project.

Creating and Using Pin Maps

A pin map allows you to define how DUT (device under test) pins and sites connect to instruments and instrument channels. A single pin map can support multiple instrument types and multiple sites, enabling you to test at scale.
Measurement Plug-In supports pin maps and uses information from a pin map file to initialize and manage NI driver sessions.

Use InstrumentStudio to create or edit a pin map. InstrumentStudio provides a graphical experience (shown below) for creating and editing pin maps, which are implemented in XML.



For more information, refer to ***Setting an Active Project Pin Map*** in the InstrumentStudio user manual.

Managing Pin Map Files

NI recommends saving all related files for each DUT you will test within the same folder. The folder should contain your InstrumentStudio project, the pin map for the DUT, TestStand files, and any other files you will use to test and measure the DUT.

Before you can use a pin map to take a measurement, the pin map must be active. Refer to ***Accessing Pin Maps in Measurement Logic*** for more information.

Related reference:

- Accessing Pin Maps in Measurement Logic
- Pin Map Contents

Related information:

- Setting an Active Project Pin Map
- Applying a Pin Filter
- Pin Map File XML Structure (TSM)

Pin Map Contents

A test system typically consists of NI hardware with inputs and outputs that connect to a DUT (device under test). Pin maps describe the hardware, DUT, and connections present within your test system. This encoded description enables NI software to take measurements based on your physical system configuration. Refer to the section below for detailed descriptions of items commonly used in pin maps.

- ***Instrument*** — software representation of equipment such as oscilloscopes, frequency generators, or digital multimeters. You must add each instrument used in the test system to the pin map.

  > **Note** When using third-party equipment, you must add a ***custom instrument*** to your pin map.

- ***Channel*** — Input or output connection point to a data acquisition system or to an ***instrument***. For example, you would use a value between 0 and 7 to specify a single channel on a PXIe-5105 oscilloscope, which corresponds to the 8 available channels on this device.
- ***Pin*** — Physical input or output of a device you are testing. There are two pin types to select from in the Pin Map Editor:
  - ***DUT pin*** — means one of the following:
    - A specific pin on the DUT (device under test).

        - A resource on the tester or DIB (device interface board) that connects to an ***instrument*** and is associated with one or more ***sites***. A resource can have one ***connection*** per ***site*** or one ***connection*** per group of ***sites***.
    - ***System pin*** — A resource on the tester or DIB that connects to an ***instrument***. A ***system pin*** is a single ***connection*** associated with all ***sites***.
- ***Site*** — Physical location of a DUT. When testing multiple instances of the same DUT at multiple sites, you must use similar connections for each site.

  Note Sites are numbered automatically starting at ***0***. You cannot assign custom values to sites.

- ***Connection*** — Pin map representation of the connection between an ***instrument channel*** and a ***pin*** on a ***site***, or the connection between an ***instrument channel*** and a ***system pin***.

Editing Pin Maps

Refer to the following sections for information about changing the contents of your pin map.

Adding Items to a Pin Map

Use the **Pin Map** tab of the **Pin Map Editor** to create a pin map entry and specify attributes for each item you add. Complete the following steps to add an item to a pin map.

1. Click **<Add [...] Here>** to display the interactive pane for this item type.
2. Click the button of the item you want to add to the pin map.
3. Specify attributes for the new item using the fields in the panel.
4. (Optional) Use the **Comment** button to document additional information about the new item.

Refer to ***Driver Instrument Reference*** for more information on how to configure instrument attributes.

Pin Map Connections

Pin map connections enable you to specify routing between instrument channels and DUTs (devices under test). Refer to the following section for information on configuring pin map connections.

Note NI recommends using the **Connections Table** in the Pin Map Editor to configure connections settings. This view shows all available routing options in your pin map.

Connections Table

The connections table displays all the defined connections within a pin map. Use the table to see an overview of all defined connections within a pin map, and filter connections for specific views.

Accessing the Connections Table

Select the top-level **Connections** section within the Pin Map tab of the Pin Map Editor to display the connections table.

Filtering the Connections Table View

Use the **View Connections For:** drop-down to filter connections based on the desired connection type.

Changing Connection Configurations

The connections table enables you to edit connection settings for all defined connections. Use the drop-down controls in each row to change connections for each pin map connection.

Note You do not have to connect to every pin in your pin map, but you might receive a warning message for unconnected pins.

Driver Instrument Reference

Refer to the following sections for information about attributes associated with specific instrument types.

General Information

- The value specified for the **Name** attribute must match the device alias. This alias can be found in NI MAX.
- All instrument panels contain a drop-down menu for specifying the instrument type. Use this to change the selected instrument to another type.
- Use the **Cut**, **Copy**, and **Paste** buttons - or select these options from the right-click menu- to perform those actions on selected instruments.

  Note Use the **Delete** key on your keyboard to remove instruments.

- Incorrectly specified attribute values may cause the Pin Map Editor to generate an error message. If possible, identify the source of the error, and refer to the ***Notes*** column for the instrument type below for potential fixes.

DCPower

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXI1Slot3`	Must match the device alias, which can be found in NI MAX.
Number of Channels	Number	`1`	This number should match the available number of channels on your device.
Channel Group Name	Text	• `CommonDCPowerChannelGroup` • `SMUSite3`	Used for channel expansion. Each name defines a session. Using

Attribute Name	Data Type	Example value	Notes
			the same group name for multiple device/ channel combinations adds them to the same session.
Channel(s)	Text	`0,1,2,3`	Values must be within the range of the number of channels on the device. In this example, the device has 4 channels.

Note Measurement Plug-In does not support DCPower sessions that do not contain a channel group.

Digital Pattern

Table 2. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXI1Slot3`	Must match the device alias, which can be found in NI MAX.
Number of Channels	Number	`32`	This number should match the available number of channels on your device.

Attribute Name	Data Type	Example value	Notes
Group	Text	• `CommonDigitalPatternChannelGroup` • `DigPatSite3`	Used for channel expansion. Each name defines a session. Using the same group name for multiple devices adds them to the same session.

DMM

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXI1Slot3`	Must match the device alias, which can be found in NI MAX.

SCOPE

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXI1Slot3`	Must match the device alias, which can be found in NI MAX.
Number of Channels	Number	`2`	This number should match the available number of channels on your device.

Attribute Name	Data Type	Example value	Notes
Group	Text	• `CommonScopeChannelGroup` • `ScopeSite3`	Used for channel expansion. Each name defines a session. Using the same group name for multiple devices adds them to the same session.

FGEN

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXI1Slot3`	Must match the device alias, which can be found in NI MAX.
Number of Channels	Number	`1`	This number should match the available number of channels on your device.

DAQmx

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`Voltage Rails`	This attribute specifies the DAQmx task name.
Channel List	Text	`Dev1/ai0,Dev1/ai8`	List of channels associated with a task. You must use the format shown in the example - the instrument name and channel name are separated by a forward slash, and qualified

Attribute Name	Data Type	Example value	Notes
			channels are separated by commas.

Custom Instrument

Table 11. Instrument Attributes

Attribute Name	Data Type	Example value	Notes
Name	Text	`PXISlot3`	For instruments using VISA and instrument drivers, use the VISA resource name or alias.
Instrument Type ID	Text	`VisaDmm`	Must match the Type ID specified in the Measurement code.

Running a Measurement from InstrumentStudio

You can run Measurement Plug-In measurements from InstrumentStudio.

1. Open the measurement in InstrumentStudio.
2. On the Home screen, select **Manual Layout**.
   The Edit Layout screen appears.
3. Locate your measurement plug-in in the list. Select **Create large panel** in the drop-down list for your measurement plug-in.

   > **Note** If your measurement plug-in does not appear in the Edit Layout window, ensure that the plug-in is either configured to start automatically or has been started manually. For more information, refer to ***Statically Register a Measurement Service*** section of the ***Developing a Measurement Plug-In with LabVIEW*** or ***Developing a Measurement Plug-In with Python*** topics.

   The large panel updates to list the selected measurement plug-in.
4. Click **OK**. InstrumentStudio opens a new project and displays the measurement UI.
5. **Optional:** Ensure that the project contains any referenced pin map and that a pin map is marked as active.
6. Click **Run** in the measurement UI. The measurement executes and values update in the measurement UI.

   > **Tip** You can change the behavior of the **Run** button so that the measurement runs continuously. Click the arrow at the side of the button and select **Run Continuously** to change the button behavior. Once the measurement is running, the button text changes. Click **Stop** to stop the measurement.

Related concepts:

- Understanding Measurement Plug-In Behavior

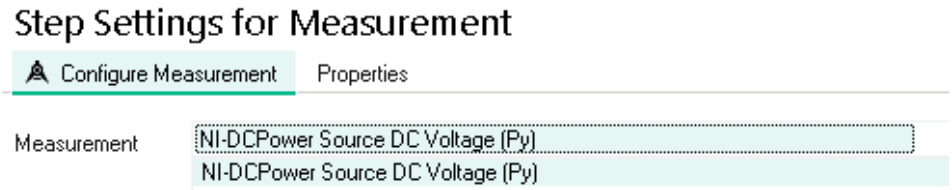
Related tasks:

- Creating and Using Pin Maps

Running a Measurement from TestStand

You can run Measurement Plug-In measurements withTestStand.

1. Add the measurement to a TestStand sequence.
   a. Launch TestStand.
   b. Double-click the **Measurement** step in the Insertion Palette. A measurement step is added to your sequence.
   c. Select the new measurement step in the sequence and select the **Configure Measurement** tab for the step.
   d. Select your measurement in the **Measurement** list.



   Measurement parameters appear in the Configure Measurement tab.

> **Note** If your measurement does not appear in the Measurement list, ensure that the plug-in is either configured to start automatically or has been started manually. For more information, refer to ***Automatically Starting a Measurement Plug-In***.

   e. Review and modify the measurement parameters.
2. If the measurement is pin-aware, configure the sequence to use your pin map. See the Using a Pin Map to Enhance Automation section for more information.
3. Run the sequence. The measurement runs as a step in your sequence.

Using a Pin Map to Enhance Automation

Pin maps contain information about connected instruments. Measurement Plug-In uses information from a registered pin map to determine the associated driver for NI hardware. You can then use the Measurement Plug-In session management service to manage instrument driver sessions for your test sequence.

Complete the following steps to register your pin map:

1. Double-click the **Update Pin Map** step in the Insertion Palette. An update pin map step is added to your sequence.

   > **Note** For best performance, register your pin map in the ProcessSetup sequence callback or use a separate Setup step group. Avoid registering a pin map within the Main sequence.

2. Select the **Update Pin Map** step in the sequence and select the **Update Pin Map** tab to configure the pin map path.
3. Specify the **Pin Map Path**. Type the path directly or click **Browse** and select the pin map file.

Using the Session Manager Service

Once you have registered a pin map, you can configure a sequence to initialize and close driver sessions for use with your measurement. Refer to Using Driver Sessions in TestStand for detailed instructions.

> **Note** LabVIEW and Python Measurement Plug-In releases include example measurements that demonstrate how to use NI hardware with the session management and pin map services. Many Measurement Plug-In examples include TestStand sequences to demonstrate pin map and session handling.

Related concepts:

- Understanding Measurement Plug-In Behavior

Related tasks:

- Using Driver Sessions in TestStand
- Creating and Using Pin Maps

Using Driver Sessions in TestStand

The session management service manages instrument driver sessions that are associated with the pin map. Because this action relies on the pin map, you must first register your pin map with the pin map service.

Use the ProcessSetup and ProcessCleanup sequence file callbacks to efficiently initialize and close driver sessions for the steps that run as part of your MainSequence sequence. In the ProcessSetup sequence instrument sessions are reserved, opened, registered, and then unreserved. You must unreserve the sessions to make them available to individual measurements. In the ProcessCleanup sequence all registered sessions are reserved, closed, unregistered, and finally unreserved.

Note Many Measurement Plug-In examples include TestStand sequences to demonstrate pin map and session handling. Some examples use step groups instead of the callback sequences described in this topic. For those examples, the Setup step group corresponds to the ProcessSetup sequence callback and the Cleanup step group corresponds to the ProcessCleanup sequence callback.

Accessing Sequence File Callbacks

Complete the following steps in TestStand to open the Sequences list and ensure the appropriate sequence file callbacks are visible.

1. Select **View** » **Sequence File** » **Sequences** to open the Sequences tab.
2. In the Sequences tab, right-click and select **Sequence File Callbacks**. The Select Callbacks dialog box appears.
3. In the list of callbacks, select the **ProcessSetup** and **ProcessCleanup** callbacks.
4. Click **OK**. The selected callbacks are listed in the Sequences tab.
5. Click a sequence to select it for editing.

Creating the ProcessSetup Sequence

In the ProcessSetup sequence, you must register your pin map with the pin map service. The Measurement Plug-In session manager service relies on pin map information to manage sessions and communicates directly with the pin map service to access a registered pin map and determine session requirements.

Use the **Update Pin Map** custom step to register the pin map with the pin map service. Complete the following steps to register your pin map:

1. Double-click the **Update Pin Map** step in the Insertion Palette. An update pin map

step is added to your sequence.
2. Select the **Update Pin Map** step in the sequence and select the **Update Pin Map** tab to configure the pin map path.
3. Specify the **Pin Map Path**. Type the path directly or click **Browse** and select the pin map file.

Complete the following steps to instantiate sessions for use by individual measurement steps:

1. In a code module in the ProcessSetup sequence file callback, call the ReserveSessions() method to reserve all sessions. Specify an empty string for the pin_or_relay_names and instrument_type inputs to return session information for all pins connected to resources in the registered pin map file.
2. Open the instrument sessions and detach the sessions. Use the session information from the previous step to call the Initialize Measurement Plug-In APIs to open sessions for each instrument.
   - In LabVEW, call Attach gRPC Session followed by Close Sessions.
   - In Python, initialize the sessions with the initialization behavior set to `ATTACH_TO_SESSION_THEN_CLOSE`.

   Note You must generally use grpc-enabled NI driver APIs to create driver sessions that can be shared between measurement services.
3. Call the RegisterSessions() method to signal to the session management service that you have initialized the driver sessions. This will enable future calls to the Session Management API to provide information to you about whether the sessions are currently open, and will allow you to later request a list of all open sessions.
4. Call the UnreserveSessions() method to unreserve the sessions so that they can be reserved by individual measurement steps.

Creating the ProcessCleanup Sequence

Complete the following steps to close sessions once the test sequence completes:

1. In a code module in the TestStand ProcessCleanup sequence file callback, call the ReserveAllRegisteredSessions() method to retrieve a list of open instrument sessions and reserve those sessions for closure.

2. Close and detach the instrument sessions. You can iterate through the list of driver sessions and use the Measurement Plug-In API to close each session.
   - In LabVEW, call Attach gRPC Session followed by Close Sessions.
   - In Python, initialize the sessions with the initialization behavior set to `ATTACH_TO_SESSION_THEN_CLOSE`.
3. Call UnregisterSessions() to unregister the sessions.
4. Call the UnreserveSessions() method to unreserve the sessions so that they are free for other clients.

Monitoring or Debugging Measurements

You can use InstrumentStudio to interactively debug a measurement step while it runs in an automated TestStand sequence. Measurement Plug-In also provides copy and paste buttons so that you can copy parameter values between different instances of the same measurement. Note that only the calling application may control a running measurement.

Debugging Tips

- You can use breakpoints with measurement steps in TestStand to debug your sequence as you would with other step types.
- Opening the measurement in InstrumentStudio allows you to run the measurement interactively, or to run the measurement from TestStand and monitor the measurement UI behavior in InstrumentStudio.
- Starting your measurement service from a development environment temporarily displaces any statically registered measurement service with the same service class. When you stop the service, the statically registered instance is restored upon being called. This behavior is useful for rapid debugging. Ensure a unique service class for each measurement service to avoid unexpected behavior.
- While a measurement service runs, there are multiple ways to execute the associated measurement logic. The measurement UI toolbar indicates where active measurement data originates.

Note To ensure monitoring is enabled, open your sequence in TestStand. In the Variables pane, set **FileGlobals** » **Measurement Plug-In** » **EnableMonitoring** to `True`. You must enable monitoring to ensure InstrumentStudio can monitor a measurement running in TestStand. Enabling monitoring may impact measurement performance.

Debugging Tools

Use the buttons next to the **Measurement** list to reload values, to open the measurement in InstrumentStudio, or to copy and paste parameter values between InstrumentStudio and TestStand and between measurements within the same

application.

Table 8. Measurement Plug-In monitor and debugging toolbar buttons

Control	Description
	In TestStand, reloads the measurement parameter values and updates the measurement UI. This button also refreshes the measurements list to display added or removed measurements.
	In TestStand, opens the measurement UI in InstrumentStudio. Note the following behaviors: • If the measurement is running in TestStand, InstrumentStudio opens the measurement with existing parameter values and pin map associations. • In other scenarios, including when the measurement sequence is open but not running in TestStand, you may be required to do the following before InstrumentStudio exhibits expected behavior: ◦ Enable monitoring by editing the Measurement Plug-In FileGlobals variables in TestStand. ◦ Manually associate a pin map with the measurement in InstrumentStudio. ◦ Copy and paste values from TestStand using the associated toolbar buttons.
	Copy measurement parameter values.
	Paste measurement parameter values.

Displacing Statically Registered Measurement Services

You can temporarily displace a statically registered measurement service. This behavior is useful for rapidly debugging measurement logic.

When you statically registered your measurement plug-in, the Measurement Plug-In discovery service starts and stops your measurement service as needed. If you start

your measurement service from a development environment using the same service class as the statically registered measurement service, the Measurement Plug-In discovery service detects the new instance of the service and stops the statically registered instance. When the development instance of the service stops, the Measurement Plug-In discovery service restarts the statically registered instance as needed.

In an application environment, be sure to specify unique service classes to avoid unexpected behavior.

Plug-In Library

A ***plug-in library*** is an HTTP web API that contains services you use to take measurements. Plug-in libraries enable you to share and collaborate across teams.

Plug-In Library Installation

Windows Installation

The plug-in library is available as a stand-alone service, or as an optional package when installing InstrumentStudio using NI Package Manager.

To install the stand-alone service, search for **Plug-In Library** in NI Package Manager.

Note To install the optional command-line interface (CLI), search for **Plug-In Library CLI** in NI Package Manager.

To use a plug-in library with InstrumentStudio, select the optional **Plug-In Library Service** package when installing InstrumentStudio in NI Package Manager.

Note The plug-in library CLI installs by default with InstrumentStudio.

For more information on installing NI software, refer to the ***NI Package Manager*** user manual.

Linux Installation

NI Linux Device Drivers are required to use the plug-In library service and optional CLI on Linux. Refer to ***Installing NI Drivers and Software on Linux Desktop*** for more information on installing required drivers and feeds.

Use the following package names to install the plug-in library service and CLI:

- `ni-plugin-library-service-bin`
- `ni-plugin-library-cli-bin`

By default, the plug-in library service (`nipluginlibrary.webapi`) requires root access to run. If you do not want to run the service as root or via `sudo`, you can add a user to the `nipluginlib` group. This group owns files and folders accessed by the plug-in library service.

Related information:

- NI Package Manager User Manual
- Installing NI Drivers and Software on Linux Desktop

Starting a Plug-In Library

To start a plug-in library, run the `nipluginlibrary.webapi` binary. This file is located in the following paths:

- Linux: Type (`nipluginlibrary.webapi`) and press **Enter** to locate the file.
- Windows: `<Program Files>\National Instruments\PluginLibrary`

Once you have started the plug-in library, it will listen to requests from any source on port `43100` by default. These settings can be modified by following the instructions in ***Plug-In Library Settings***.

Plug-In Library Settings

The following sections describes how to customize plug-in library behavior.

Changing Maximum Upload Size

To specify the allowed size of services being uploaded to a plug-in library, enter the following command:

```
nipluginlibrary.webapi.exe --PLUGINLIBRARY:MAX_UPLOAD_SIZE=<bytes>
```

- To specify bytes, enter the number value, with no suffixes.
- To specify megabytes, use either `MB` or `mb` as a suffix.
- To specify gigabytes, use either `GB` or `gb` as a suffix.

Note To set this value using environment variables, create the following

variable:

```
PLUGINLIBRARY__MAX_UPLOAD_SIZE=<bytes>
```

Changing the Default Host and Port

To change the plug-in library host and port address, enter the following command:

```
nipluginlibrary.webapi.exe --URLS=http://<host>:<port>
```

If you want to change the default port, enter the following command:

```
nipluginlibrary.webapi.exe --HTTP_PORTS=<port>
```

- The `<host>` value specifies where web requests can come from. The default value for this parameter is `*`. This value allows requests to come from anywhere. To restrict requests to the local host, use `localhost`.
- The `<port>` value specifies the port used. To assign ports dynamically, use `0`.

Note You cannot use `localhost` when specifying dynamic port binding. Instead, use either `127.0.0.1` or `[::1]` to specify dynamic port binding for locally restricted requests.

Assigning Port and Host Addresses via Environment Variables

Create the following environment variables to configure settings for your plug-in library:

- To specify the host and port address, use `URLS=http://<host>:<port>`
- To specify only the port address, use `HTTP_PORTS=<port>`

Dynamic port that accepts requests from anywhere

```
nipluginlibrary.webapi.exe --URLS=http://*:0
```

Single port that only accepts local requests, with a mamixum upload size of 300 MB

```
nipluginlibrary.webapi.exe --URLS=http://localhost:43102
        --PLUGINLIBRARY:MAX_UPLOAD_SIZE=300mb
```

Plug-In Library Configuration Files

Plug-in library settings can be configured using configuration files. Refer to the following sections for information on how to create and use configuration files to manage plug-in library settings.

Configuration File Format

The following code snippet shows the typical contents of a `.JSON` configuration file.

```
{
  "URLS": "https://localhost:43207",
  "PLUGINLIBRARY":{
    "MAX_UPLOAD_SIZE": "300mb"
  }
}
```

Configuration File Name and Location

You must name your configuration file `pluginlibrary.json`.

Configuration files must be saved at the following paths, depending on your OS:

- Windows: `%localappdata%\National Instruments\PluginLibrary\pluginlibrary.`
- Linux: `$HOME\.config\ni\pluginlibrary.json`

Using a Plug-In Library

Once you have created a plug-in library, you can interact with it using the command line interface (CLI). The following sections describe how to make changes to a plug-in

library using CLI commands.

> **Note** To access the command line interface, open a terminal and enter `nipluginlib`.

Adding a Service to a Plug-In Library

Complete the following steps to add a new service to a plug-in library.

Enter the following command: `nipluginlib publish path --url`
The following example shows how to use the `publish` command:

```
nipluginlib publish C:\services\ai\single_channel --exclude .venv/**/* --force -u
http://pluginlib.mycompany.com:43100
```

Where:

- `--exclude` specifies to not publish any files in `.venv` to the library.
- `--force` instructs the server to over-write the existing service in the library.
- `-u` specifies the server URL.

> **Note** Either `-u` or `--url` can be used to specify the library URL.

Removing Services from a Plug-In Library

Complete the following steps to remove a service from the plug-in library using the CLI.

Before removing a service from a library, you must know its ID. This information can be found by using the `list` command. See ***Listing Plug-In Library Contents***.
Enter the following command: `nipluginlib delete id --url`.

Listing Plug-In Library Contents

Complete the following steps to view plug-in library contents via the CLI.

Enter the following command: `nipluginlib list --url`

Plug-In Versioning

Creating versioned measurement plug-ins enables development and deployment alignment, as well as traceability. Plug-in versioning is supported in TestStand and InstrumentStudio. The following sections describe how to implement versioning with your developed plug-ins.

Versioning Plug-Ins Using Python

Complete the following steps to add version numbering to your Python-developed measurement plug-in.

Note You must use Python SDK version 2.1.0 or newer to version plug-ins.

1. Open the service configuration file (`.serviceconfig`) for the measurement plug-in you want to version.
2. Specify the new version number in the the `version` field.
3. Remove the `version` parameter from any instances of the `MeasurementService` class.

Note Plug-ins developed with Python do not auto-increment versions. You must repeat this process each time you create a new version of a plug-in.

Versioning Plug-Ins Using LabVIEW

This section describes how to implement versioning for LabVIEW-developed measurement plug-in.

Implementing Versioning when Developing LabVIEW Measurement Plug-Ins

To implement versioning for your measurement plug-in, open `Get Measurement Details.vi` within your measurement plug-in library (`.lvlib`) and specify the version number in the `version` field in the `measurement details` cluster.

Note You must enter the version number in the diagram view in LabVIEW. Changes made to the VI front panel will not be applied to the plug-in.

Note Modifying the version number within the Version Information section of your plug-in build specification will not change the plug-in version. This version information only applies to the build itself.

Incrementing Plug-In Versions using LabVIEW

To create a new version of a measurement plug-in, open the `Get Measurement Details.vi`, select the diagram view, and enter the new version number in the `version` field in the `measurement details` cluster.

Using Measurements in Custom Applications

Measurement Plug-In uses the cross-platform, open-source gRPC framework to maximize performance and interoperability. You can use gRPC calls to interact with your measurement logic while your measurement is running as a service. Deploy your measurement plug-in to allow the Measurement Plug-In discovery service to automatically start your measurement service.

Related concepts:

- Understanding Measurement Plug-In Behavior

Related tasks:

- Statically Register a Measurement Service

Related information:

- gRPC Framework Website (grpc.io)

Measurement Development and Usage Best Practices

This topic describes patterns that may be useful as you develop or use a measurement.

- **Cancellation**—Not all drivers handle cancellation in the same way.
  - For drivers that support cancellation or asynchronous abort, you must still handle the gRPC cancellation and forward it to the driver API.
  - For drivers that do not support cancellation or asynchronous abort, break long API calls and waits into shorter iterations and handle cancellation events between iterations.
- **Progressive UI Updates**—For best performance with continuously running measurements, maximize the time between UI measurement data updates. Continuous, rapid UI updates while a measurement runs may reduce measurement performance.
- **Python Decorators**—Decorators must be listed in the same order as the measurement function signature parameters and return values.
- **Session Management**—

  For best performance, use the Measurement Plug-In API to manage sessions whenever possible. Consult the examples for the latest Measurement Plug-In release to understand how to structure calls to the API and efficiently configure a measurement in TestStand. The following text describes general principles and best practices associated with session management.

  Calling the ReserveSessions() method instructs the session management service to block access to a session. If you reserve an instrument session, you must unreserve it before it is accessible to other callers.

  - Be sure to handle error cases where an error occurs after a session is reserved but before the session is used.
  - Make use of the TimeoutInMilliseconds parameter.
  - Do not reserve sessions for pins that your measurement does not use.

  Calling the RegisterSessions() method indicates that the instrument driver session

is open. This information is used by other callers to determine if they should close the instrument session when they are done with it.

- Reserve a session before you attempt to register or unregister it.
- Do not register a session if you did not open it.
- Do not close a session if you did not register the session.

In general, you should not register or unregister sessions within your measurement logic. Instead, you should normally register and unregister sessions as part of the setup or cleanup phases of your execution framework. For example, refer to Using Driver Sessions in TestStand.

The session management service depends on the NI gRPC Device Server (gRPC-device). Once a gRPC-device session is opened it is cached and subsequent open calls do not create new sessions. Closing a gRPC-device session closes the underlying sessions. For best performance, reuse sessions whenever possible rather than creating new sessions.

- **TestStand Best Practices**—

  For best performance, use the Measurement Plug-In API to manage sessions whenever possible. Consult the examples for the latest Measurement Plug-In release to understand how to structure calls to the API and efficiently configure a measurement in TestStand. The following text describes general principles and best practices associated with session management. Consider the following practices and tips while working with sessions in TestStand.

  - Reserve, open, and register all instrument sessions in the ProcessSetup sequence to initialize sessions once and reduce overhead for individual measurements in the test sequence.
  - Specify the InstrumentTypeId parameter when calling ReserveSessions() or ReserveAllRegisteredSessions() to limit reserved sessions to a specific instrument type.
  - Do not use ReserveAllRegisteredSessions() before opening and registering instrument sessions.

  Measurement steps in TestStand cache measurement service connections. If you restart the measurement service while the sequence is open, you may invalidate cached connections and produce a networking error. If a measurement step returns a networking error (such as "connection refused"), the following actions

may clear the error:

- Select **File** » **Unload All Modules**.
- Refresh the measurement parameters.

Enabling monitoring in TestStand may impact measurement performance. Do not enable monitoring unless it is necessary for your application.

- **Log Files**—Log files can be found in `C:\ProgramData\National Instruments\MeasurementPlugin\Logs`.

Supported Datatypes

Measurement Plug-In supports the following datatypes:

- Signed integers
  - Int32
  - Int64
- Unsigned integers
  - UInt32
  - UInt64
- Floating-point numbers
  - Float (32-bit floating-poing number)
  - Double (64-bit floating-point number)
- Boolean
- String
  - Pin (string type specialization)
  - Path (string type specialization)
- 1D array (supported for all base datatypes and type specializations)
- Enum
- XY data

API Reference

Refer to the Related Information section on this page to access API reference documentation for Measurement Plug-In.

Related information:

- Measurement Plug-In

Measurement Plug-In Architecture and Data Flow

The topics in this section describe individual Measurement Plug-In services, relevant data flow interactions, and other considerations.

Understanding the Discovery Service

The Measurement Plug-In discovery service provides a registry of other services, and can discover and activate other services on the system. These features allow the discovery service to distinguish, manage, and describe measurement services on the system.The Measurement Plug-In discovery service runs automatically when InstrumentStudio or TestStand enumerate available measurement services. To ensure that the discovery service is running, follow the initial steps for adding and running a measurement in either InstrumentStudio or TestStand.

Registering a Measurement Service with the Discovery Service

Activating a measurement service requires a `*.serviceconfig` file which includes information describing the service. Services that register a `*.serviceconfig` file must call RegisterService() when the service starts or registration will never succeed when the discovery service attempts to start the measurement service.

Enumerating Registered Measurement Services

Call the EnumerateServices() method to return the list of registered measurement services.

Related concepts:

- Understanding Measurement Plug-In Behavior

Related tasks:

- Statically Register a Measurement Service

Related information:

- Discovery Service Proto File

Understanding the gRPC Device Server Activation Service

The Measurement Plug-In gRPC Device Server Activation Service ensures that NI gRPC Device Server is available for Measurement Plug-In. When you resolve the service class ni.`ni_measurement_plugin_sdk`.v1.grpcdeviceserver, Measurement Plug-In gRPC Device Server Activation Service launches NI gRPC Device Server with a dynamic port assignment to avoid collision with any existing instances of NI gRPC Device Server. The port number is then registered with the Measurement Plug-In Discovery Service.

Identifying the NI gRPC Device Server Port

To determine the NI gRPC Device Server port, query the Measurement Plug-In Discovery Service using the ni.`ni_measurement_plugin_sdk`.v1.grpcdeviceserver service class and one of the provided interfaces listed in the `GrpcDeviceServerActivationService.serviceconfig` file:

- nidaqmx_grpc.NiDAQmx
- nidcpower_grpc.NiDCPower
- nidigitalpattern_grpc.NiDigital
- nidmm_grpc.NiDmm
- nifgen_grpc.NiFgen
- niscope_grpc.NiScope
- niswitch_grpc.NiSwitch

You can also manually edit the `server_config.json` file found in the same directory as `NationalInstruments.MeasurementPlugin.GrpcDeviceServerActivationService.exe` to specify a static port assigment.

The default location for both the `GrpcDeviceServerActivationService.serviceconfig` and `server_config.json` files is `<Program Files>\National Instruments\`

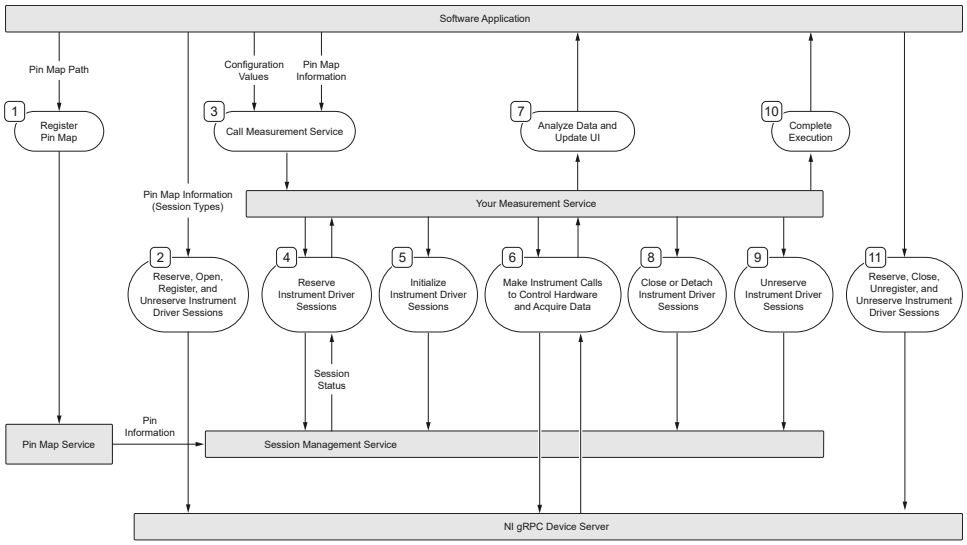
```
Shared\Plug-Ins\Services\gRPC Device Server Activation\.
```

Understanding the Driver Session Management Service

The session management service manages driver sessions and driver session access. Use driver or channel information from a pin map to specify the appropriate sessions for NI driver software. This allows you to write pin-aware measurements and avoid directly addressing I/O channels. The driver session management service tracks driver session registration to ensure that only one measurement service accesses a driver session at a time.

This section describes the behavior and data flow for session management in . The API simplifies session management, and should be used whenever possible to manage sessions for supported NI driver software.

Figure 1. Pin Map, Session, and Instrument Call Data Flow



1. The software application registers the pin map with the pin map service.
2. The software application calls the session management service to instantiate required instrument driver sessions. The application reserves, opens, registers, and finally unreserves the driver sessions.
3. The application calls your measurement service, which executes the measurement logic.
4. The measurement service calls the session management service to reserve a required driver session. For GPIB, serial, or LXI instruments, use NI-VISA to perform session management. The session management service returns session

information, including session status.

5. The measurement service initializes instrument driver sessions, either by creating new sessions on the NI gRPC Device Server or attaching to an existing session.
6. Your measurement service makes instrument calls via NI instrument drivers that support gRPC.
    1. If a driver does not provide a gRPC interface you can use gRPC directly to communicate with your instrument. In Python, use the `grpcio` module. In LabVIEW, use grpc-labview (available in github). Refer to the examples for NI gRPC Device Server for additional implementation details.
    2. For GPIB, serial, or LXI instruments, use NI-VISA to control your instrument.
7. Your measurement service performs data analysis and updates the measurement UI.
8. The measurement service closes or detaches instrument driver sessions.
9. Your measurement service calls the session management service to unreserve the instrument driver session once the measurement completes.
10. Your measurement service completes execution and returns values to the software application.
11. The software application reserves, closes, unregisters, and unreserves the instrument driver sessions.

Controlling Access to Driver Sessions

The Measurement Plug-In session management service provides a reservation mechanism to ensure that only one measurement at a time has access to each driver session. This prevents measurements from changing the state of an instrument that is in use by another measurement.

The Measurement Plug-In session management service reserves each driver session when a measurement retrieves driver session information via the ReserveSessions() or ReserveAllRegisteredSessions() methods. If a driver session is already reserved by another measurement, these methods wait until either the driver session becomes unreserved or the timeout expires.

Accessing Open Driver Sessions

The Measurement Plug-In session management service ensures that only one measurement at a time has access to each driver session. This prevents measurements

from changing the state of an instrument that is in use by another measurement.

To attach to a driver session within your measurement logic, first ensure that you have sessions registered with the Session Management service. Refer to Using Driver Sessions in TestStand for more information.

Within your measurement logic, use the ReserveSessions() method to access the driver session information.

Some measurements must use the session_exists Boolean to determine whether to perform one-time setup.

- **NI-DAQmx**—Create or add channels to the task only when initializing a new task on the server (and session_exists is `False`).
- **NI-Digital Pattern Driver**—Loading files (for example, specifications, patterns, or pin maps) into the session will produce an error if the files were previously loaded. Checking that session_exists is `False` is a good way to avoid loading files multiple times. You can also explicitly unload files.

Table 9. Boolean value descriptions for session_exists

Value of session_exists	Description	Notes
`false`	The driver session has not been initialized.	Your measurement logic should initialize (and later close) the driver session.
`true`	The driver session has been initialized.	Your measurement logic should not attach or detach to the driver session.

Be sure to call the UnreserveSessions() method when your measurement is finished using a driver session. You must call this method even when an exception or error causes your measurement to exit to ensure that sessions are not reserved indefinitely. If you are using LabVIEW, the Measurement Plug-In LabVIEW API handles this requirement automatically.

If a measurement service service crashes or closes while it has reserved a driver session, subsequent reservations will trigger the specified reserve session timeout behavior. In order to recover, you must restart the discovery service as well as any manually-launched measurements.

Supply a TimeoutInMilliseconds parameter to a reserve method to specify a behavior while waiting for the reservation to succeed:

Table 10. Reserve Session Timeout Behavior

Value	Behavior
-1	Specifies no timeout. The measurement will wait indefinitely until the session can be reserved.
0 (default)	Specifies that an exception occurs immediately if the session cannot be reserved.
Any positive numeric value	Specifies a timeout, in milliseconds, after which an exception occurs if the session cannot be reserved.

Sharing Driver Sessions between Multiple Measurement Services

This topic describes generally how a measurement service should connect to an instrument session depending on the instrument or instrument driver in use.

Measurement Plug-In supports session management for a subset of NI driver software. The following figure lists those drivers and their support for gRPC, including:

- Whether a driver is supported by NI gRPC Device Server
- Whether a driver provides a native gRPC interface for a supported software language

Use the criteria in this table to determine the suitability of any driver for your application. Drivers which do not provide native gRPC support must make explicit gRPC calls to the NI gRPC Device Server.

Table 11. gRPC support in target drivers (as of 2024 Q1, earliest supported driver versions listed for each feature)

Supported NI driver software	Driver supported by NI gRPC Device Server	Driver version implements native gRPC interface (Python)	Driver version implements native gRPC interface (LabVIEW)
NI-DAQmx	✔	2023 Q3	–
NI-DCPower	✔	2023 Q1	2023 Q1

Supported NI driver software	Driver supported by NI gRPC Device Server	Driver version implements native gRPC interface (Python)	Driver version implements native gRPC interface (LabVIEW)
NI-Digital Pattern Driver	✔	2023 Q1	2023 Q2
NI-DMM	✔	2023 Q1	2023 Q2
NI-FGEN	✔	2023 Q1	2023 Q2
NI-SCOPE	✔	2023 Q1	2023 Q2
NI-SWITCH	✔	2023 Q1	–
NI-VISA (GPIB, serial, LXI interfaces)	✔ (as of 2024 Q1)	–	–

NI-DCPower, NI-Digital Pattern Driver, NI-DMM, NI-FGEN, NI-SCOPE, NI-SWITCH

Measurement services must make gRPC calls to the NI gRPC Device Server in order to create or attach to a session, perform the measurement, and destroy or detach from the session. Use the Measurement Plug-In API to handle session management calls for these drivers.

1. In Python, the driver API will make the gRPC calls for you if you specify the grpc_options parameter for the Session constructor.
2. In LabVIEW, the driver API will make the gRPC calls for you if you use the gRPC-specific Initialize/Attach VIs.

NI-DAQmx

Measurement services should make explicit gRPC calls to the NI gRPC Device Server in order to create or attach to a task, perform the measurement, and destroy or detach from the task.

NI-VISA

Instruments connected via GPIB, serial, or LXI interfaces can use NI-VISA for session management. Registering the session ensures that only one measurement service

communicates with the instrument at any given time. For an example, refer to the ***Keysight 34401A DMM Measurement*** example files.

Understanding the Pin Map Service

The pin map service retrieves information from a registered pin map. This information is used by the session manager service. Do not call into the pin map service directly (except to register or unregister the pin map). Instead, use the session manager service, which wraps around the pin map service. The pin map service can retrieve lists of pins, pin groups, or relays; session or resource names; or instrument channels for a pin.

Note Activating a pin map in software does not interact with the pin map service. Instead, activating a pin map allows the software application to specify the pin map to be used with a measurement.

Accessing Pin Maps in Measurement Logic

When a pin map is configured properly in InstrumentStudio or TestStand, those applications will pass a PinMapContext object to your measurement service. This object contains an identifier for the pin map as well as a target site number. The PinMapContext object is a required input for many session management API methods.

Using Pin Maps with Measurements in InstrumentStudio

To register a pin map for a measurement in InstrumentStudio, ensure that the project contains an active pin map. Refer to the related information section for more information about setting an active pin map in InstrumentStudio.

Using Pin Maps with Measurements in TestStand

Refer to ***Using Driver Sessions in TestStand*** for more information about setting an active pin map in TestStand.

Related tasks:

- Using Driver Sessions in TestStand

Related information:

- Setting an Active Project Pin Map

Measurement Plug-In Pin Map Service: Supported Instruments

Refer to the Related Information section on this page to access the Measurement Plug-In Pin Map service `*.proto` file, which lists supported instruments.

Related information:

- Pin Map Service Proto File