



FlexLogger Plug- in Development Kit



FlexLogger Plug-in Development Kit

Contents

- 1.1 Welcome to the FlexLogger Plug-in Development Kit User Manual 4
- 1.2 What is the FlexLogger Plug-in Development Kit? 6
 - 1.2.1 Key Features 6
 - 1.2.2 Components of the FlexLogger Plug-in Development Kit 6
- 1.3 FlexLogger Plug-in Development Kit Requirements 8
- 1.4 New Features and Changes 10
 - 1.4.1 FlexLogger Plug-in Development Kit 2024 Q3 10
 - 1.4.2 FlexLogger Plug-in Development Kit 1.7 10
 - 1.4.3 FlexLogger Plug-in Development Kit 1.6 10
 - 1.4.4 FlexLogger Plug-in Development Kit 1.5 10
 - 1.4.5 FlexLogger Plug-in Development Kit 1.4 10
 - 1.4.6 FlexLogger Plug-in Development Kit 1.3 10
 - 1.4.7 FlexLogger Plug-in Development Kit 1.2 10
 - 1.4.8 FlexLogger Plug-in Development Kit 1.1 11
- 1.5 Software Version Compatibility 12
- 1.6 Installing the FlexLogger Plug-in Development Kit 14
- 1.7 Creating a FlexLogger Plug-in from a Template 16
 - 1.7.1 FlexLogger Plug-in Examples 18
- 1.8 FlexLogger Plug-in File Structure 20
 - 1.8.1 Loading FlexLogger Plug-ins from Custom Locations 21
- 1.9 Using Plug-ins in FlexLogger 22
 - 1.9.1 Plug-in and Channel Parameters 23
- 1.10 FlexLogger Plug-in Development 26
 - 1.10.1 Accessing the FlexLogger Palette in LabVIEW 28
 - 1.10.2 FlexLogger Plug-in LabVIEW Classes 29
 - 1.10.2.1 Processing Element Class 30
 - 1.10.2.2 Plug-in Class 33
 - 1.10.2.3 Channel Class 34
 - 1.10.2.4 System Interface Class 48
 - 1.10.2.5 Additional Classes 48
- 1.11 FlexLogger Plug-in Debugging 50
 - 1.11.1 Debugging Plug-in Loading Errors 50
 - 1.11.2 Viewing Errors in FlexLogger 50
 - 1.11.3 Interactively Testing Plug-ins 51

1.11.3.1Interactive Debugging Session Example 56

Welcome to the FlexLogger Plug-in Development Kit User Manual

The FlexLogger Plug-in Development Kit User Manual provides detailed descriptions of the product functionality and the step-by-step processes for use.

Looking For Something Else?

For information not found in the User Manual for your product, such as specifications and API reference, browse *Related Information*.



RELATED INFORMATION

[*Download the FlexLogger Plug-in Development Kit*](#)

[*License Setup and Activation*](#)

[*FlexLogger Plug-in Development Kit Release Notes*](#)

[*FlexLogger User Manual*](#)

[*FlexLogger Python API Overview*](#)

[*FlexLogger Python API Reference*](#)

What is the FlexLogger Plug-in Development Kit?

The FlexLogger Plug-in Development Kit is a FlexLogger add-on designed for test engineers. Use the FlexLogger Plug-in Development Kit to create LabVIEW plug-ins that perform custom in-line calculations and acquire data from NI hardware supported by FlexLogger and third-party hardware.

Key Features

- Easy integration with FlexLogger
- Plug-in templates to:
 - communicate with NI hardware supported by FlexLogger and third-party hardware
 - perform custom in-line calculations
 - generate waveform data

Components of the FlexLogger Plug-in Development Kit

The FlexLogger Plug-in Development Kit is an optional add-on designed for use in a FlexLogger test system.

- FlexLogger
- LabVIEW
- LabVIEW Application Builder Module

FlexLogger Plug-in Development Kit Requirements

The FlexLogger Plug-in Development Kit works with LabVIEW and FlexLogger. Refer to the LabVIEW documentation and FlexLogger documentation to ensure your system meets the specified minimum requirements.



RELATED INFORMATION

[*FlexLogger System Requirements*](#)

[*LabVIEW System Requirements*](#)

New Features and Changes

Learn about updates—including new features and behavior changes—introduced in each version of the FlexLogger Plug-in Development Kit.

FlexLogger Plug-in Development Kit 2024 Q3

- Added support for LabVIEW 2024

FlexLogger Plug-in Development Kit 1.7

- Added support for LabVIEW 2023 Q1

FlexLogger Plug-in Development Kit 1.6

- Send strings from third party devices to FlexLogger. Use the Produce Data for FlexLogger plug-in template and configure the Write Data VI to write string channel data.



RELATED INFORMATION

[Writing Channel Data from a Plug-in to FlexLogger](#) on page 35

FlexLogger Plug-in Development Kit 1.5

- Added support for LabVIEW 2022

FlexLogger Plug-in Development Kit 1.4

- Added support for LabVIEW 2021

FlexLogger Plug-in Development Kit 1.3

- Added support for LabVIEW 2020

FlexLogger Plug-in Development Kit 1.2

- Support for digital input plug-in channels

FlexLogger Plug-in Development Kit 1.1

- New Perform calculation plug-in template type
 - Send FlexLogger analog input channels
 - Publish calculation results back to FlexLogger with the same timing information as the mapped channels
- Custom plug-in commands offer users a new way to interact with plug-ins

Software Version Compatibility

Determine which versions of FlexLogger and LabVIEW are compatible with your FlexLogger Plug-in Development Kit.

Table 1 : Compatibility information for the FlexLogger Plug-in Development Kit, FlexLogger, and LabVIEW

FlexLogger Plug-in Development Kit version	FlexLogger version	LabVIEW version
2024 Q3	2024 Q3 or later	2024 (64-bit)
1.7	2023 Q3 or later	2023 (64-bit)
1.6	2023 Q2 or later	2022 (64-bit)
1.5	2023 Q1 or later	2022 (64-bit)
1.4	2022 Q2 or later	2021 (64-bit)
1.3	2020 R3 or later	2020 (64-bit)
1.2	2020 R2 or later	2019 (64-bit)
1.1	2019 R4 or later	2019 (64-bit)
1.0	2019 R3 or later	2019 (64-bit)

Installing the FlexLogger Plug-in Development Kit

Install the FlexLogger Plug-in Development Kit using NI Package Manager.

Before you begin

Before you begin, refer to *Software Version Compatibility* to check which versions of LabVIEW, FlexLogger, and the FlexLogger Plug-in Development Kit are compatible.

About this task

Use NI Package Manager to download and install LabVIEW, FlexLogger, and the FlexLogger Plug-in Development Kit. For more information, refer to *Installing, Updating, Repairing, and Removing NI Software* in the Package Manager documentation.

Procedure

1. Install LabVIEW.
2. Install FlexLogger.
3. Install the FlexLogger Plug-in Development Kit.



RELATED INFORMATION

[Software Version Compatibility](#) on page 12

Determine which versions of FlexLogger and LabVIEW are compatible with your FlexLogger Plug-in Development Kit.

[Installing, Updating, Repairing, and Removing NI Software with Package Manager](#)

Creating a FlexLogger Plug-in from a Template

About this task

Use the FlexLogger Plug-in Wizard and plug-in templates to create plug-ins that you can use in FlexLogger.

Procedure

1. Launch LabVIEW.
2. In LabVIEW, select **File»Create Project**.
3. From the **Create Project** window, select **FlexLogger IO Plug-in** and click **Finish**.
4. In the **Create FlexLogger IO Plug-in** window, update the **IO Plug-in Name** and select a **Plug-in Template** based on your goal.

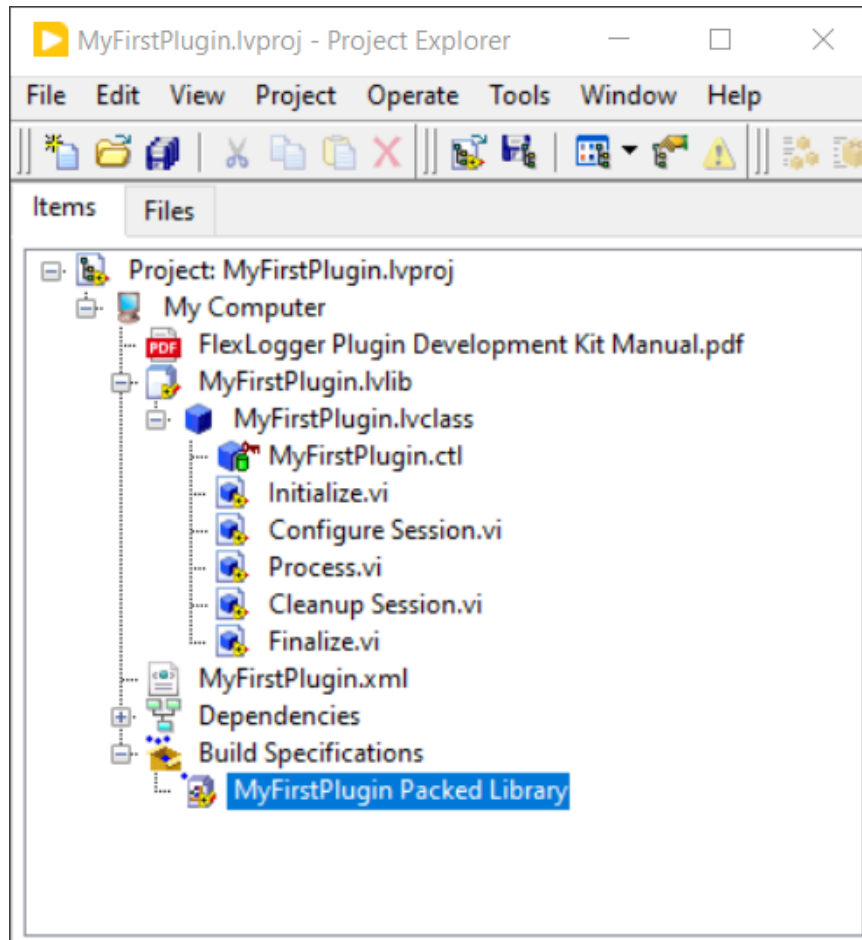
Table 2 : FlexLogger Plug-in Templates

Goal	Plug-in Template	Data Flow
Create a plug-in that sends a setpoint value you enter in FlexLogger to a plug-in channel. Control a third-party instrument with FlexLogger. For more information, refer to <i>Read Setpoint Channels from FlexLogger</i> .	Consume setpoint data	User input → plug-in
Create a plug-in that consumes one or more waveforms from FlexLogger channels, performs a custom calculation, and produces a waveform of resulting data that can be logged in FlexLogger.	Perform calculation	FlexLogger → plug-in and plug-in → FlexLogger
Create a plug-in that generates waveform data and sends it to FlexLogger.	Produce data for FlexLogger	plug-in → FlexLogger

5. Optional: Update the **Description** and **Destination Directory (Source)** for the plug-in. By default, the wizard saves new plug-ins in the `My Documents\LabVIEW Projects\FlexLogger IO Plug-ins\` directory.
6. Click **Create**.

The FlexLogger Plug-in Wizard creates a LabVIEW project with necessary dependencies, a new LabVIEW class, an XML file, and a packed library build specification.

7. Optional: Customize your plug-in using the methods described in the *Plug-in Development* section.
8. To build your plug-in, select the Packed Library build specification and click **Build Selected**.



FlexLogger Plug-in Wizard builds the packed library file and XML file for the plug-in and saves them in the default FlexLogger plug-in directory, %public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins.

What to do next

After creating a plug-in, use it in your FlexLogger project.



RELATED INFORMATION

[Read Setpoint Channels from FlexLogger](#) on page 38

[FlexLogger Plug-in Development](#) on page 26

Learn about the components of a plug-in so you can customize a plug-in you created from a template or develop a new plug-in.

[Using Plug-ins in FlexLogger](#) on page 22

Add and configure plug-in channels to your FlexLogger project so you can perform custom in-line calculations or collect data from third-party hardware devices.

FlexLogger Plug-in Examples

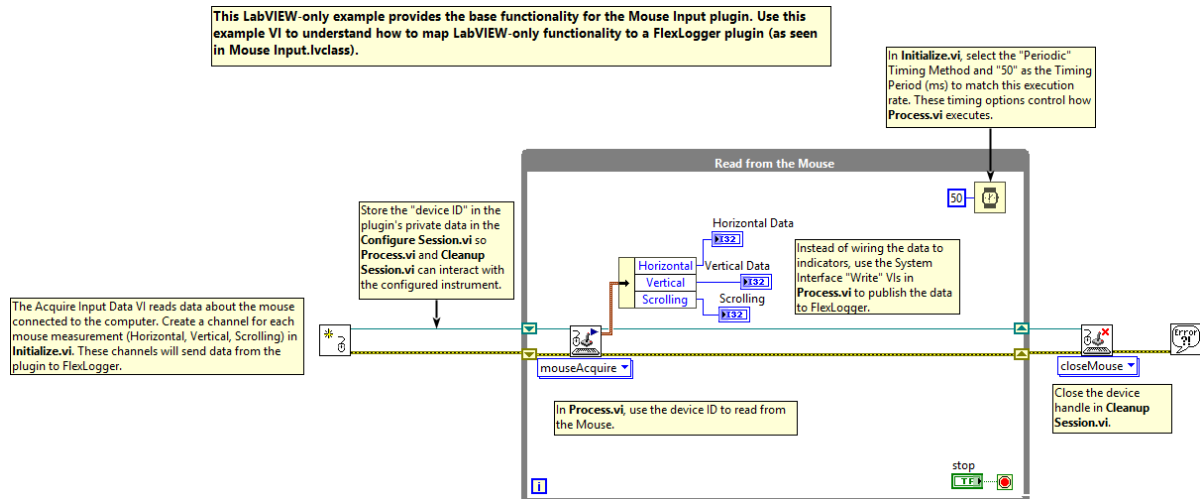
Learn about the various examples included in the FlexLogger Plug-in Development Kit. The examples are located in the %ProgramFiles%\National Instruments\LabVIEW <version>\examples\FlexLogger\IO Plugins directory.

Table 3 : FlexLogger Plug-in Examples

Example	Type	Description
Cycle Counter	Perform calculation	Counts the number of times a specified channel rises above a threshold
Digital Port Calculator	Perform calculation	Combines eight individual digital channels into a single analog value representing the port value
IVI DC Power Supply	Consume setpoint data	Uses the IVI DC Power class driver to control a power supply instrument To load this IVI example, first install the IVI Compliance Package from NI Package Manger and an IVI DC Power-compliant instrument driver to control the power supply.
IVI Meter – DC Voltage	Produce data for FlexLogger	Reads DC voltage values from an IVI DMM class driver-compliant instrument To load this IVI example, first install the IVI Compliance Package from NI Package Manger and an IVI compliant instrument driver to read voltage values from an instrument.
Mouse Input	Produce data for FlexLogger	Reads data from the mouse connected to the computer
Rosette	Perform calculation	Calculates a rosette measurement from multiple strain input channels

LabVIEW-only VIs

Some examples include a LabVIEW-only VI that demonstrates how to map functionality from a standalone VI to a FlexLogger plug-in class. For example, the LabVIEW project for the Mouse Input example contains **Mouse Input - LabVIEW Only Example VI** that describes how different parts of the block diagram correspond to the VIs in `Mouse Input.lvclass`.



FlexLogger Plug-in File Structure

Learn where FlexLogger looks for plug-ins so you can load the plug-ins you created.

When you build a plug-in using the FlexLogger Plug-in Wizard, the wizard saves the packed library file and XML file for the plug-in in the default FlexLogger plug-in directory, %public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins. FlexLogger loads plug-ins from this default directory. You can configure FlexLogger to [load plug-ins from custom locations](#).

A plug-in named MyFirstPlugin, for example, has the following file names and locations.

Table 4 : FlexLogger Plug-in File Locations

Folder	%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins\MyFirstPlugin
Packed library file	%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins\MyFirstPlugin\MyFirstPlugin.lvlibp
XML file	<p>%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins\MyFirstPlugin\MyFirstPlugin.xml</p> <p>FlexLogger uses the XML file to register and load the plug-in for use at run-time. The XML file contains the plug-in name, description, and version. If you update the plug-in or change the plug-in name, you must manually update the XML file.</p>



NOTE

- The folder, packed library file, XML file must all have the same name.
- FlexLogger does not support multiple plug-ins with the same name.
- When you modify a plug-in, you must close and re-open your FlexLogger project to use the updated plug-in.

Versioning

When you create a plug-in, the FlexLogger Plug-in Wizard assigns a version number to the plug-in. The wizard defines the version number in the <Version> and <OldestCompatibleVersion> tags in the plug-in XML file. Update the <Version>

and `<OldestCompatibleVersion>` values to track updates to a plug-in that you have already added to a FlexLogger project.



RELATED INFORMATION

[Loading FlexLogger Plug-ins from Custom Locations](#) on page 21

Configure FlexLogger to load plug-ins from locations outside of the default directory.

Loading FlexLogger Plug-ins from Custom Locations

Configure FlexLogger to load plug-ins from locations outside of the default directory.

About this task

By default, FlexLogger loads plug-ins from `%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins`. Complete the following steps to load plug-ins from additional, custom locations.

Procedure

1. Using a text editor, create a file that specifies additional plug-in locations using the following JSON format.


```
{ "AdditionalPluginPaths": [ "C:\\Users\\<username>\\Documents\\Flex Logger Projects\\", "<another absolute path>" ] }
```
2. Save the file in the root directory of `%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins`. FlexLogger does not have specific requirements for the file name.
3. Change the file extension from `.txt` to `.config`.
4. Re-launch FlexLogger.

Results

The plug-ins you saved in the custom file locations are visible when you select **Add channels»Plug-in** in the **Channel Specification** toolbar.

Using Plug-ins in FlexLogger





Add and configure plug-in channels to your FlexLogger project so you can perform custom in-line calculations or collect data from third-party hardware devices.





Before you begin

If you do not have a FlexLogger project already, create a new FlexLogger project. For more information, search for *Creating a New FlexLogger Project* on ni.com/docs.

Procedure

1. Launch FlexLogger and open your project.
2. Complete the steps described below based on your goal.

Goal	Tasks
Add a plug-in to my project	<ol style="list-style-type: none"> 1. In the Channel Specification toolbar, select Add channels»Plug-in. 2. Select the plug-in you want to add. <div>  <p>NOTE If FlexLogger did not load any plug-ins on launch, the Plug-in option does not appear. For troubleshooting plug-in issues, refer to <i>FlexLogger Plug-in File Structure</i> and <i>FlexLogger Plug-in Debugging</i>.</p> </div> <p>FlexLogger adds the plug-in you selected to the Channel Specification under the local system.</p>
Configure plug-in parameters	<p>Click Configure  at the right edge of the plug-in header to open the plug-in configuration dialog box.</p> <div>  <p>NOTE When FlexLogger is running a test, you cannot modify plug-in parameters, and FlexLogger disables the plug-in Configure gear .</p> </div> <p>FlexLogger automatically generates the plug-in configuration dialog box with settings for the parameters you defined in the plug-in.</p>

Goal	Tasks
Configure channel parameters	<ol style="list-style-type: none"> 1. Hover over an individual channel to see Configure . 2. Click the gear to open the channel configuration dialog box. <p> NOTE Channel-level parameters apply to only the selected channel.</p> <p>FlexLogger automatically generates the channel configuration dialog box with settings for parameters you defined in the plug-in.</p>
Stop logging a channel	Hover over the channel and click Disable logging ().
Delete a plug-in	Click Delete  at the right edge of the plug-in header to remove a plug-in from your project.

**RELATED INFORMATION**

[FlexLogger Plug-in File Structure](#) on page 20

Learn where FlexLogger looks for plug-ins so you can load the plug-ins you created.

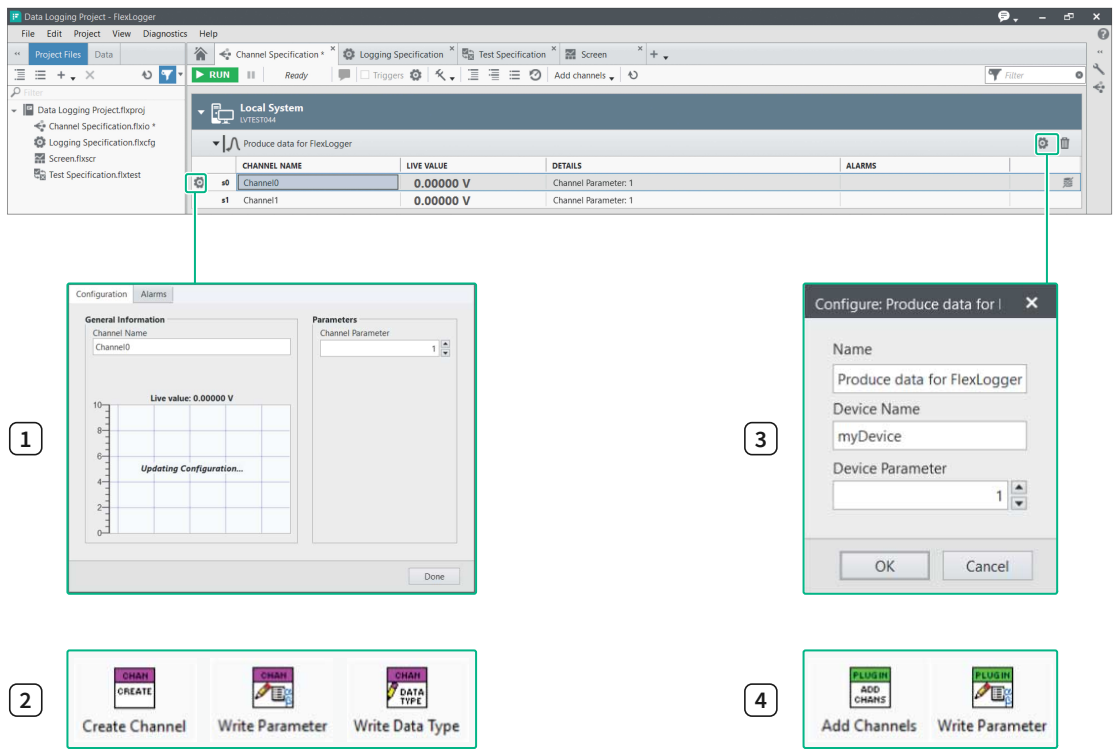
[FlexLogger Plug-in Debugging](#) on page 50

Troubleshoot plug-in problems with the debugging resources available in LabVIEW and FlexLogger.

Plug-in and Channel Parameters

Configure the plug-in and channel parameters of a plug-in to support custom calculations and data acquisition from third-party hardware devices.

Plug-in parameters, such as **Device Name**, apply to the whole plug-in. *Channel parameters* apply to a specific plug-in channel. The figure below illustrates where you can configure plug-in and channel parameters in FlexLogger and LabVIEW.



1.

FlexLogger dialog box for configuring channel parameters.
2.

LabVIEW VIs for customizing channel parameters.
3.

FlexLogger dialog box for configuring plug-in parameters.
4.

LabVIEW VIs for customizing plug-in parameters.



RELATED INFORMATION

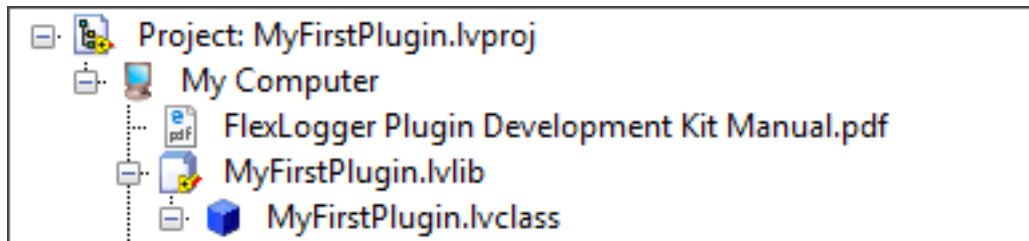
[Accessing the FlexLogger Palette in LabVIEW](#) on page 28

Use the FlexLogger palette in LabVIEW to develop and customize plug-ins.

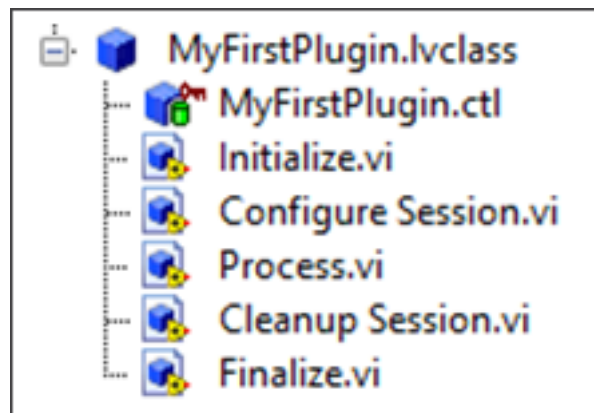
FlexLogger Plug-in Development

Learn about the components of a plug-in so you can customize a plug-in you created from a template or develop a new plug-in.

When you create a plug-in named `MyFirstPlugin`, for example, using the FlexLogger Plug-in Wizard in LabVIEW, the wizard creates a project with the hierarchy shown below.



The VIs within `MyFirstPlugin.lvclass` are the starting point for code changes. These VIs perform different roles, such as defining parameters and exchanging data with FlexLogger. For details about how these VIs function, refer to *Processing Element Class*. State information—like a hardware reference—passes between the VIs using the private class data defined in `MyFirstPlugin.ctl`.



If you plan to develop a plug-in, first test your hardware and drivers using a LabVIEW VI that communicates with third-party instruments. NI recommends that you start with the LabVIEW-only VI provided in some of the examples, including Mouse Input. This LabVIEW-only VI demonstrates how to map functionality from a standalone VI to a FlexLogger plug-in class. For more information, refer to *FlexLogger Plug-in Examples*.

Review the examples installed with the FlexLogger Plug-in Development Kit and refer to the table below for guidance on your plug-in related question.

Table 5 : Plug-in Development Guidance

Question	Where to Go
How many channels do I need?	Refer to <i>Channel Class</i> for details on how plug-in channels communicate with FlexLogger.
How many parameters will I need to edit while logging data?	Only setpoint channels can be edited while data logging is in progress. Refer to <i>Read Setpoint Channels from FlexLogger</i> for details on how to use setpoint channels.
How many plug-in parameters do I need?	Plug-in parameter values affect the entire plug-in—for example, the name of a hardware resource. Refer to <i>Plug-in Class</i> for more information.
How many channel-specific parameters do I need?	Channel-specific parameter values are unique to every channel and are defined when creating new plug-in channels. Refer to <i>Channel Class</i> for more information.
What data do I need to pass between plug-in VIs?	You must pass references to any resources you initialize or allocate. For example, if an instrument session is initialized in <i>Configure Session VI</i> , it needs to be passed to <i>Process VI</i> for use and then to <i>Cleanup Session VI</i> for disposal. Important plug-in information can be stored in the private class data of the plug-in (for example, <code>MyFirstPlugin.ct1</code>).
How do I know my plug-in works?	Use one of the following approaches to evaluate the functionality of your plug-in by executing the logic and investigating the effect of code changes. <ul style="list-style-type: none"> • Load the plug-in into FlexLogger. Refer to <i>Using Plug-ins in FlexLogger</i> for more information. • Run the plug-in with the Plug-in Environment Simulator. Refer to <i>Interactively Testing Plug-ins</i> for more information.



RELATED INFORMATION

[Processing Element Class](#) on page 30

Learn how execution states are managed by FlexLogger plug-ins.

[FlexLogger Plug-in Examples](#) on page 18

Learn about the various examples included in the FlexLogger Plug-in Development Kit. The examples are located in the %ProgramFiles%\National Instruments\LabVIEW <version>\examples\FlexLogger\IO Plugins directory.

[Channel Class](#) on page 34

[Read Setpoint Channels from FlexLogger](#) on page 38

[Plug-in Class](#) on page 33

[Using Plug-ins in FlexLogger](#) on page 22

Add and configure plug-in channels to your FlexLogger project so you can perform custom in-line calculations or collect data from third-party hardware devices.

[Interactively Testing Plug-ins](#) on page 51

Accessing the FlexLogger Palette in LabVIEW

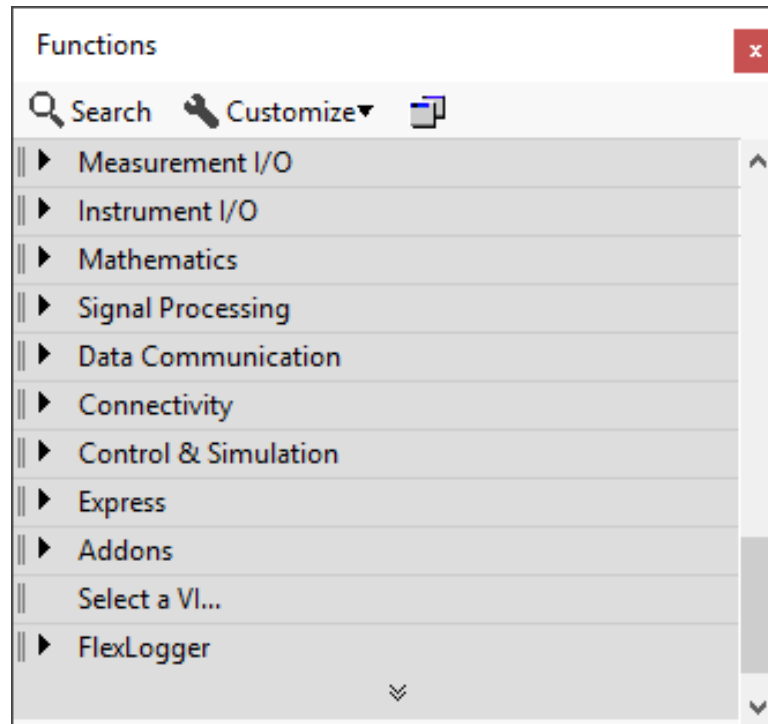
Use the FlexLogger palette in LabVIEW to develop and customize plug-ins.

About this task

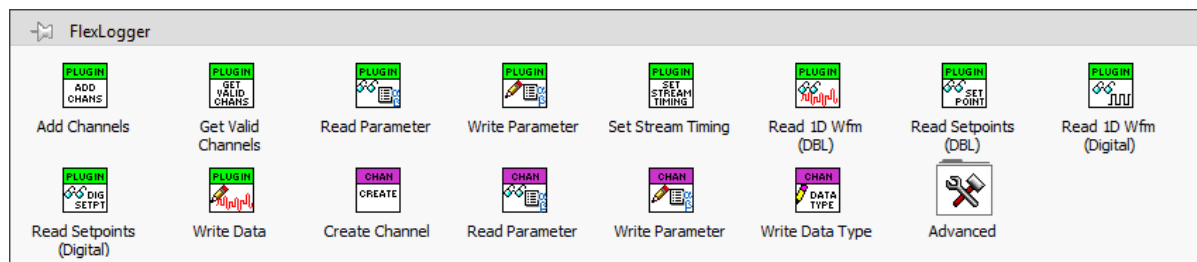
The Plug-in Development Kit installer adds the FlexLogger palette to LabVIEW. The palette provides the plug-in and channel methods needed to interact with FlexLogger.

Procedure

1. In LabVIEW, navigate to **View»Functions Palette**.



2. In the **Functions** window, select the **FlexLogger** palette.



What to do next

For more information on the methods contained in the palette, refer to *FlexLogger Plug-in LabVIEW Classes*.



RELATED INFORMATION

[FlexLogger Plug-in LabVIEW Classes](#) on page 29

FlexLogger Plug-in LabVIEW Classes

In the FlexLogger Plug-in Development Kit, `PluginSDK.lvlib` contains the classes and VIs used for plug-in development. Plug-in developers will mostly interact with the Plug-in and Channel classes.

If you are unfamiliar with LabVIEW object-oriented programming, refer to *LabVIEW Object-Oriented Programming* in the *LabVIEW Help* for conceptual and how-to documentation. To access this topic, visit ni.com/r/lvobjectprog.

Processing Element Class

Learn how execution states are managed by FlexLogger plug-ins.

The Processing Element class, `ProcessingElement.lvclass`, defines the plug-in execution states. The FlexLogger data engine manages the transitions between these states with a set of methods. Refer to the following figure and table to learn more about each method.

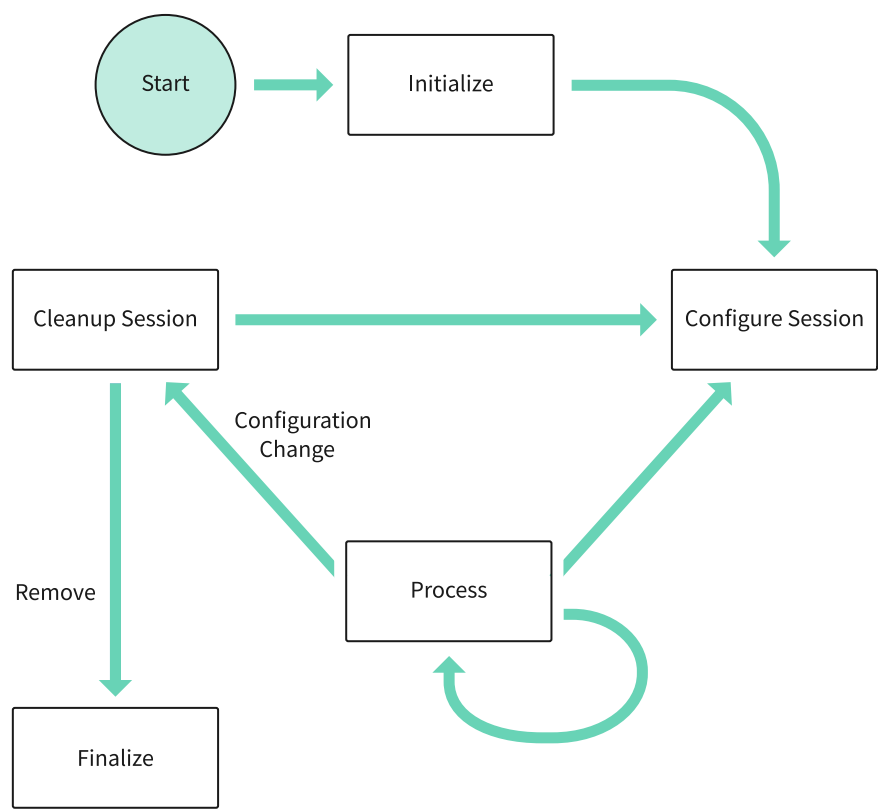


Table 6 : Processing Element Class Methods

Method	Description
Initialize	Runs only once—When you add a plug-in to the FlexLogger Channel Specification.
	Defines the timing method used by the <i>Process</i> method.
	Declare the plug-in parameters users interact with in FlexLogger.
	Declare the channels (and corresponding channel parameters) the plug-in will read data from or send data to.

Table 6 : Processing Element Class Methods (Continued)

Method	Description
Configure Session	Runs when you make a configuration change while FlexLogger is not running a test.
	Initialize and configure hardware/instrument sessions. Runs after the <i>Initialize</i> method.
	Read the latest channel parameters set in FlexLogger. Read the latest plug-in parameters set in FlexLogger.
	Set the timing information (dt, T0) for data written to FlexLogger in the <i>Process</i> method.
Process	Read what channels FlexLogger reports as valid (channels defined by the plug-in may be modified in FlexLogger).
	Read from and write to third-party hardware.
	Read channel data from FlexLogger.
Cleanup Session	Write channel data to FlexLogger.
	Runs when you make a configuration change.
Finalize	Cleans up the hardware sessions and resources initialized by the <i>Configure Session</i> method.
	Runs when you remove a plug-in from your FlexLogger project or close the project.
	Cleans up any resources initialized during the <i>Initialize</i> method. Commonly empty.

Process Method Timing

In the *Initialize* method, plug-ins which produce data or consume only setpoint data set the **Timing Parameters** that control how the *Process* method executes. If a plug-in reads waveform data from other FlexLogger channels, you should instead set the **Resampling Parameters**, which determine how input data will be resampled. Plug-ins should typically only use either the Set Plugin Timing VI or the Set Plugin Resampling VI in Initialize VI, but not both. In situations where the plug-in timing is not driven by the timing of the input data, you may want to use the Set Plugin Timing VI in addition to Set Plug-in Resampling VI.

Table 7 : FlexLogger Plug-in Process Method Timing Parameters

Timing Parameter	Timing Period (ms) required?	Description
Periodic	Yes	With a timing period of n, periodic timing ensures at least n milliseconds between runs of the Process method.
Immediate	No	Run Process without delay. This method is typically used when a blocking call inside the Process method controls timing.
On Data Ready	Optional	For use by plug-ins that read channel data from FlexLogger—such as the Consume setpoint data from FlexLogger template. On Data Ready blocks execution of Process until FlexLogger sends new channel data to the plug-in. This parameter uses a timing period of n milliseconds as a <i>time-out</i> . After passing the time-out period, the plug-in executes Process without receiving new channel data from FlexLogger. If the time-out period is ≤ 0 ms, On Data Ready blocks execution of Process until FlexLogger sends new channel data to the plug-in.
Triggered	No	Not applicable for IO Plug-in development.

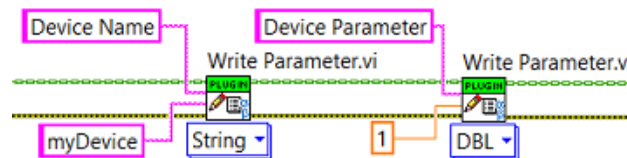
Table 8 : FlexLogger Plug-in Process Method Resampling Parameters

Resampling Parameter	Description
Resampling Mode	Determines how resampling of input waveforms will be done. In Minimum and Maximum mode the resampling rate is determined by the input channels. In Maximum mode any slower channels will be resampled to the rate of the fastest channel. In Custom mode resampling is done at a fixed rate specified by the Resampling Rate parameter. In Original Rate mode no resampling will be done. This mode should only be used if there is a single input channel.
Resampling Rate (Hz)	The rate to resample the data at if using Custom mode.
Drift Tolerance (seconds)	The amount of history to store in the plug-in waiting for time aligned data from different sources. A high number here increases the memory usage of the plug-in, a low number potentially causes issues with plug-ins which are not synchronized.
Block Size (seconds)	The minimum amount of overlapping data that should be present before the Process method is called.

Plug-in Class

The Plug-in class, `Plugin.lvclass`, is the parent class for plug-ins created with the FlexLogger Plug-in Development Kit. It provides the state management functionality for the plug-in by inheriting from `Processing Element.lvclass`. The Plug-in class also provides methods to define and interact with plug-in channels and parameters. After initializing the channels, the Plug-in class maintains channel information that correctly reflects the state of the plug-in channels in FlexLogger.

The Plug-in class supports double-precision floating point numbers, strings, enums, Booleans, and integers as plug-in parameters. Plug-in parameters can be created using the **Write Parameter VI**, as shown in the following example:

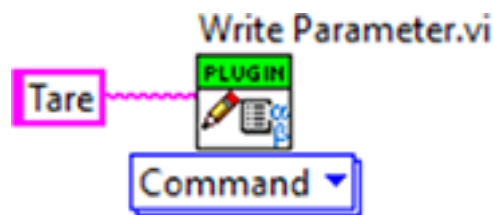


NOTE

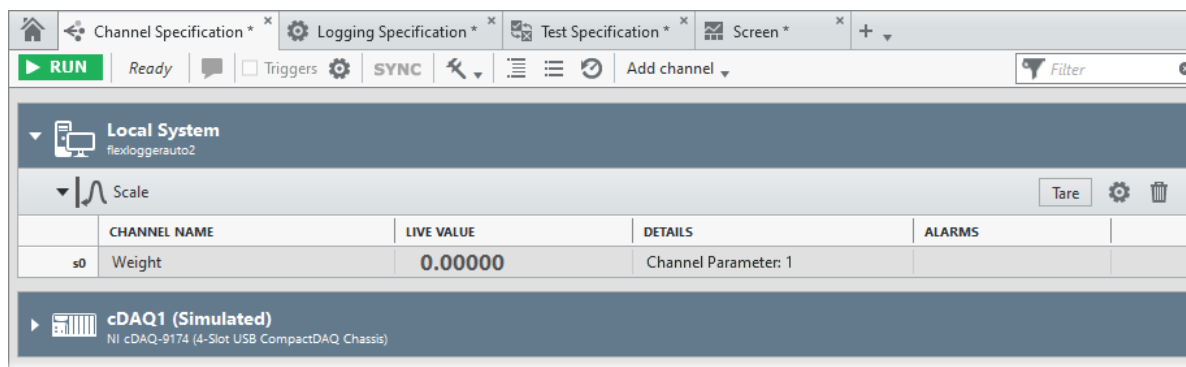
Do not call **Add Channels** multiple times from the same plug-in method as it causes parameter settings to not save correctly.

Commands

Plug-ins have a special parameter type called **Command**. When you select this parameter, the plug-in will create a button in FlexLogger. Pressing the button in FlexLogger sends a notification to the plug-in. For example, use a Command parameter to create a **Tare** button for a scale plug-in.



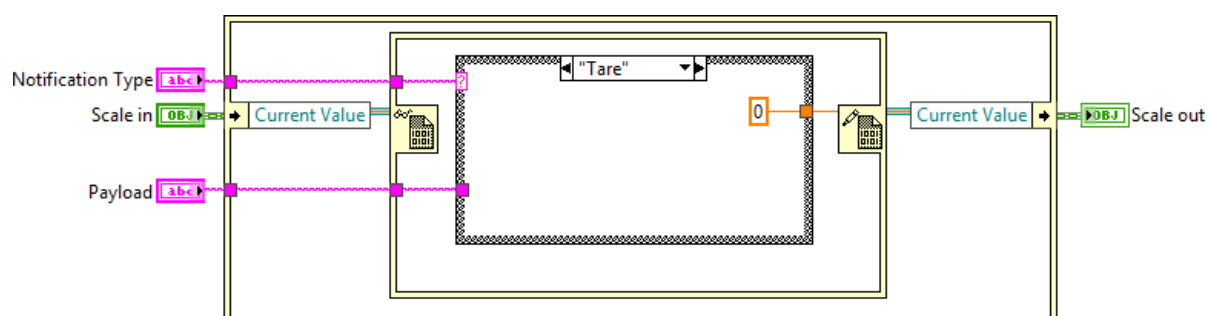
When you press the **Tare** button in FlexLogger, the plug-in tares the scale.



In order to create a user-defined command in a plug-in, you must override the `Handle Notification VI`. Create an overriding VI by right-clicking the `.lvclass` file in the project and selecting **New»VI for Override**.

This VI has two input parameters:

- **Notification Type**- a string input to match the name given to your plug-in command parameter
- **Payload**- a string input that is empty when the notification comes from a command button, except for the built-in commands, Test Start and Test Stop. The Test Start and Test Stop commands return the time (in seconds) for the start or stop of a test. Use `Notification Payload to Timestamp VI` to decode the payload.



To create a special case for Test Start or Test Stop notifications, add these notification type names to the case structure and use `Notification Payload to Timestamp VI` (`<vi.lib>\FlexLogger\SDK\PluginSDK.lvlib\PEFClasses\SDK\Plugin\`) to convert the payload to a timestamp.

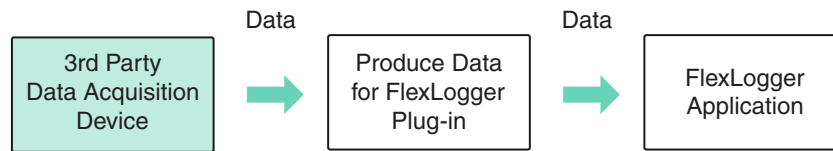
Note that `Handle Notification VI` runs in a separate loop. If you want to share data with other VIs in the Plug-in class, use a Data Value Reference instead of directly changing the private class member variables.

Channel Class

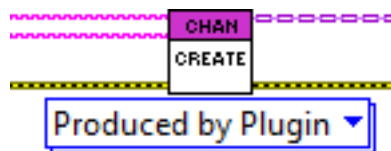
The Channel class, `Channel.lvclass`, allows plug-ins to send data to FlexLogger and read setpoint data from FlexLogger. The Plug-in class uses the Channel class to organize channels and ensure consistency between the plug-in and FlexLogger.

Writing Channel Data from a Plug-in to FlexLogger

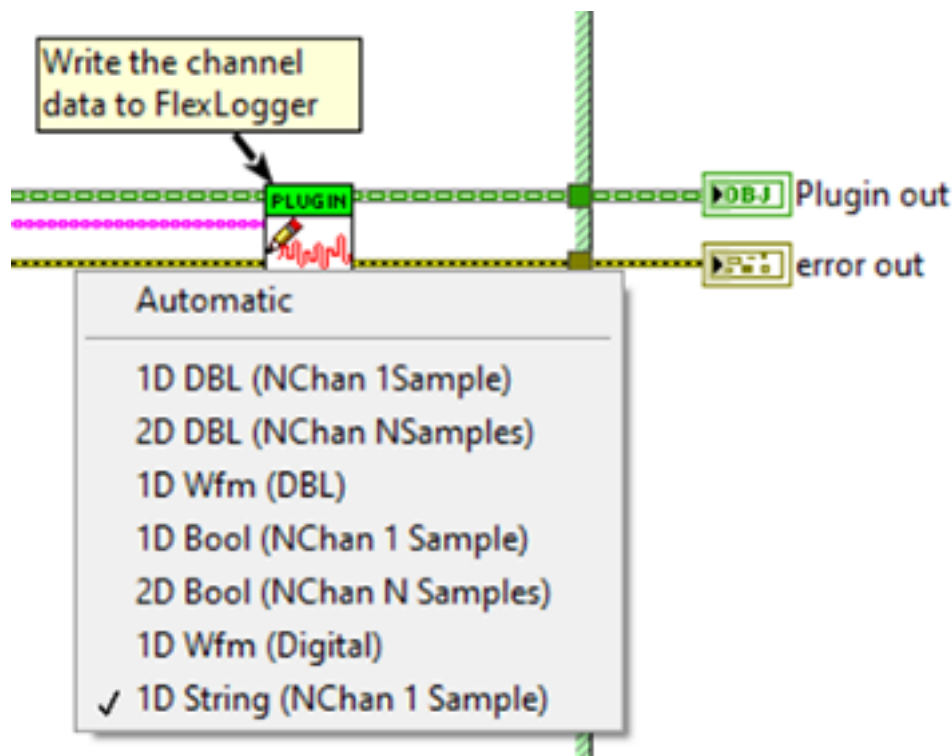
Plug-ins created from the Produce data for FlexLogger template send collected data to FlexLogger. Use the Produce data for FlexLogger template as a starting point for creating your own plug-ins that send channel data to FlexLogger.



To create a channel that sends data to FlexLogger, select the **Produced by Plugin** mode in **Create Channel VI**. The **Create Channel VI** is in the **Initialize** method of the **Produce data for FlexLogger** template.

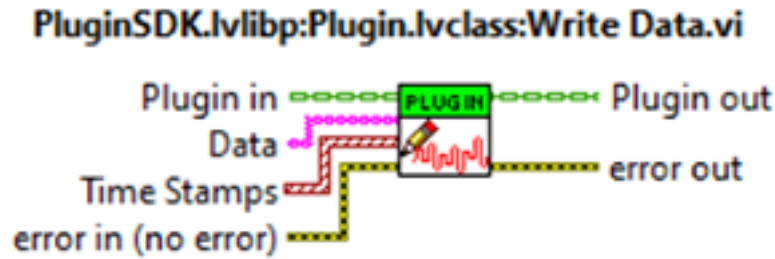


You can write string data or waveform data to a FlexLogger channel. To write string data, specify **1D String (NChan 1 Sample)** as the data type in the **Write Data VI**. The **Write Data VI** is in the **Process** method of the **Produce data for FlexLogger** template.

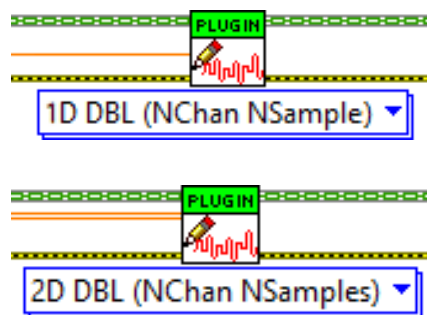


When writing strings, you can optionally specify the time stamp at which the string is written. If you don't specify a time stamp, the VI will use the current time as the time stamp

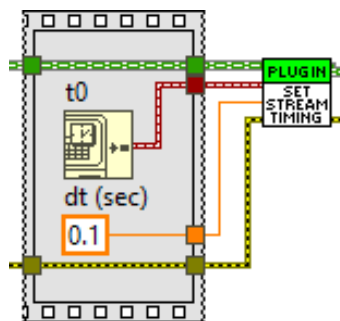
for when the string is written. The following image shows all of the inputs and outputs for the Write Data VI.



To send waveform data, specify either a one or two-dimensional array of doubles.



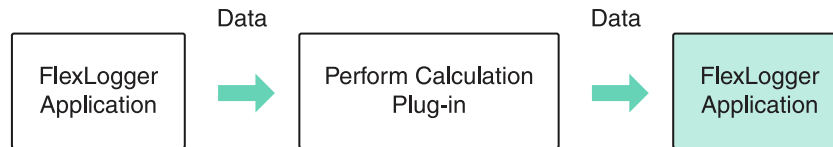
Writing waveforms that overlap in time disrupts visualization and logging capabilities of FlexLogger. The Produce data for FlexLogger template uses the Set Stream Timing VI to send data to FlexLogger without overlap errors. The Set Stream Timing VI is in the Configure Session method and defines the waveform **dt** and **Start Time** timing parameters.



Since the Configure Session method always executes before the Process method, the timing information for the output channel updates after every configuration change.

Read and Write Waveform Channels

Plug-ins created from the Perform calculation template read waveform data from FlexLogger and publish waveform or string data to FlexLogger. You can also create a plug-in which only reads waveform data and does not publish any data back to FlexLogger.



To map FlexLogger channels to a plug-in use the Plug-in class `Write Parameter VI` **Single Channel** or **Multiple Channel** modes. The **Single Channel** mode maps one named parameter to one channel while the **Multiple Channel** mode maps one named parameter to multiple FlexLogger channels.



Select the channels you want to map from the plug-in level **Configure** gear located to the right of the plug-in name.



NOTE

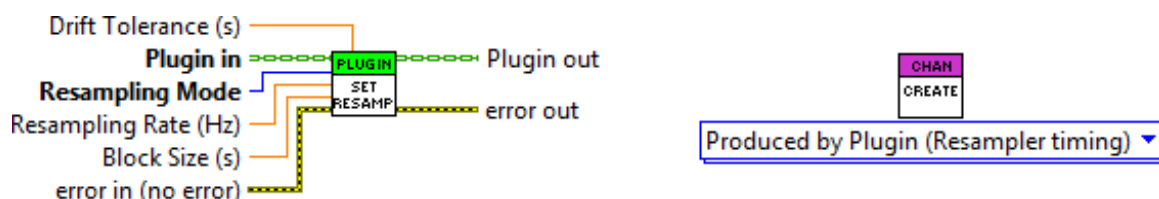
Setpoint channels do not work with waveform parameters.

You can also add support for producer channels to allow users to map other FlexLogger channels to the plug-in by using the Channel class `Write Parameter VI`.



Select the channels you want to map from the channel-in level **Configure** gear that appears to the left of the channel name when you hover over the channel.

When using the minimum or maximum resampling mode of a plug-in, it is common that the rate of the waveform channels produced by the plug-in match the rate of the input. Since the rate can change dynamically based on your channel selection in these modes, a new type of channel, **Produced by Plugin (Resampler Timing)**, is provided which automatically adapts to these changes in sample rate. NI recommends creating your producer channels with this option whenever the resampling mode is set to **Minimum** or **Maximum**.



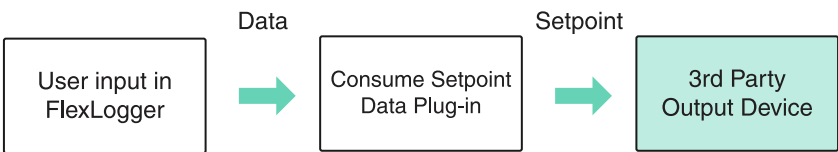


NOTE

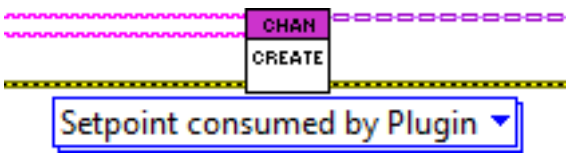
When reading waveforms from other channels, the length of the waveform will be determined by how much data that channel wrote in one cycle of its Process VI. Since the waveforms are not combined, waiting for longer periods between reads does not result in more data in the waveform, but in more waveforms in the queue that can be read by Read 1D Wfm VI.

Read Setpoint Channels from FlexLogger

Plug-ins created from the Consume setpoint data template read values from user settable controls in FlexLogger and use those values to control third-party output devices.



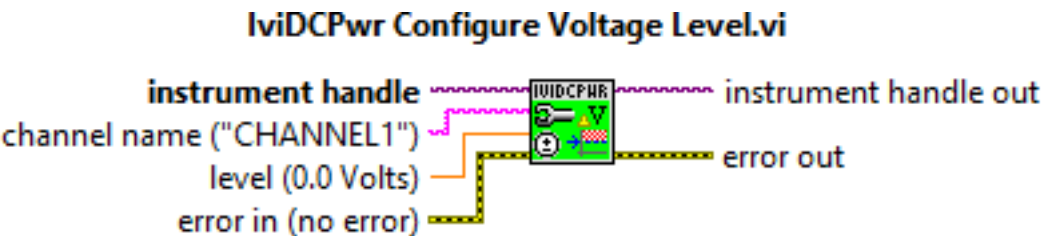
To create a channel that consumes setpoint channel information from FlexLogger, use the **Setpoint consumed by Plugin** option in Create Channel VI.



As demonstrated by the Process VI in the Consume setpoint data template, the Plug-in class Read Latest Setpoints VI extracts the most recent scalar value and provides a one-dimensional array of setpoints (one per channel).



In the IVI DC Power Supply example, the setpoint value read from FlexLogger is used as the **level** input to set the Power Supply DC voltage level.



This VI configures the DC voltage level that the power supply attempts to generate.

Retrieve Data from Individual Channels

To read waveforms from other plug-ins or use a setpoint channel in the Process VI, use either the Read 1D Wfm VI or Read Latest Setpoints VI based on the types of channels that are configured. Use Read 1D Wfm VI to return an array of data for each waveform or setpoint channel. Use Read Latest Setpoints VI to return the latest value for each waveform and setpoint channel.



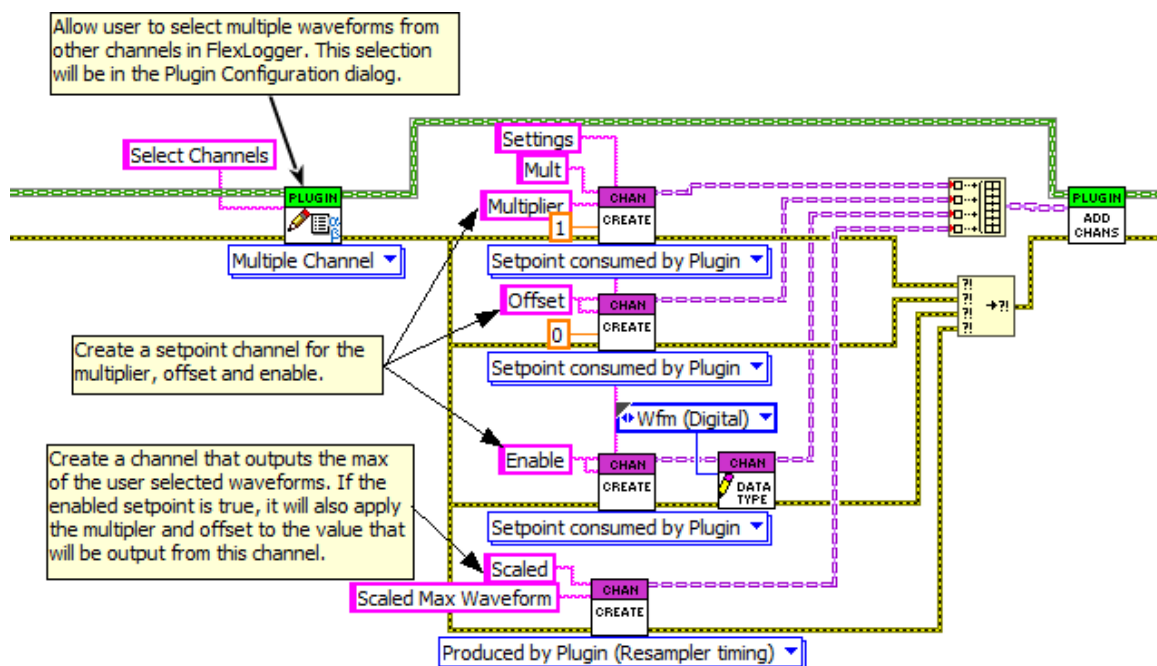
NOTE

You should only use one instance of either of these VIs in the Process VI.

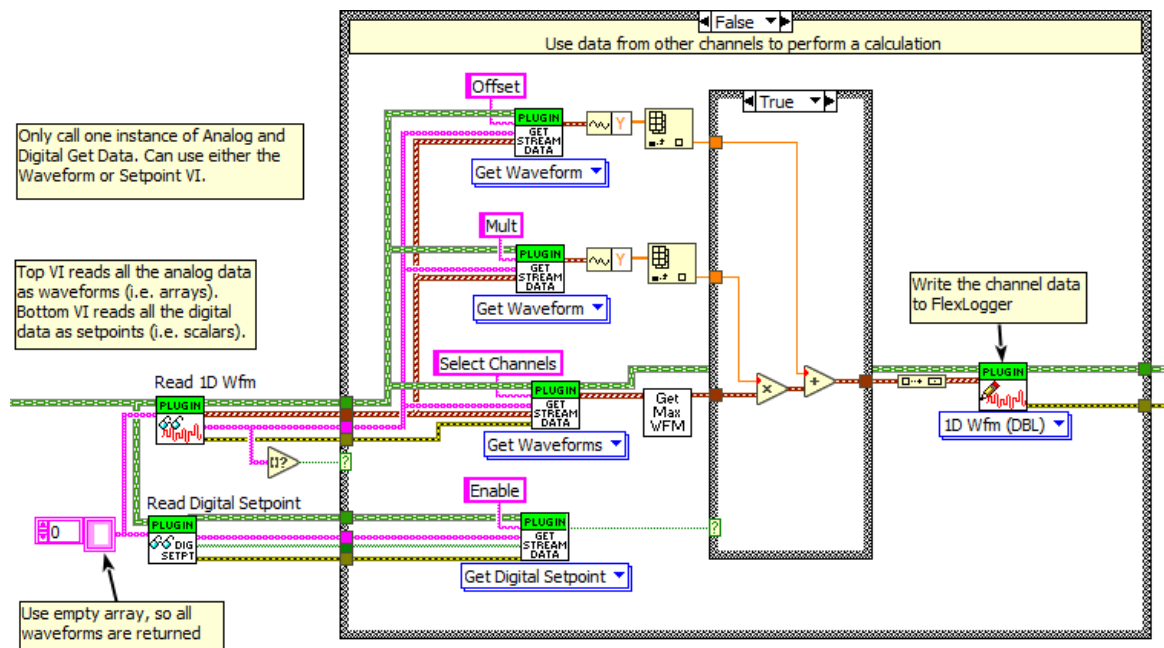
If you have a mix of analog and digital data to read, you can use one of these VIs to read the analog data and one of the corresponding digital VIs to read the digital data.

The following is an example plug-in that scales waveforms from other channels based on setpoint channels. These setpoint channels provide a multiplier, offset, and a digital setpoint to enable or disable the scaling.

The `Initialize` VI could look like the following to enable the user to select waveforms from other channels in FlexLogger and use setpoint channels to change parameters while logging data.



Based on the configuration of the previous `Initialize` VI, the `Process` VI might look like the following:



Read 1D Wfm VI gets all of the analog data and returns it as an array. The **Offset** and **Mult** parameters receive the setpoint channel data of the plug-in and **Select Channels** receives the waveform data. **Select Channels** is a plug-in parameter that references waveforms from other FlexLogger channels the user specified in the **Plug-in Configuration** dialog box. Once the data is read, the **Get Data from Stream** VI parses specific waveforms based on the unique names used when creating the channels.

Read Digital Setpoint VI gets all of the digital setpoint and waveform data and returns a single point of the most recent data for all channels.

Below is the resulting channel table in FlexLogger.

New FlexLogger IO Plugin 1 (cDAQ1Mod1/ai0, cDAQ1Mod1/ai1, cDAQ1Mod1/ai2, cDAQ1Mod1/ai3, cDAQ1Mod1/ai4)				
	CHANNEL NAME	LIVE VALUE	DETAILS	ALARMS
Scaled	Scaled Max Waveform	109.049		
▼ Settings				
	CHANNEL NAME	LIVE VALUE	DETAILS	
Enable	Enable	High		
Mult	Multiplier	10.0000		
Offset	Offset	200.000		

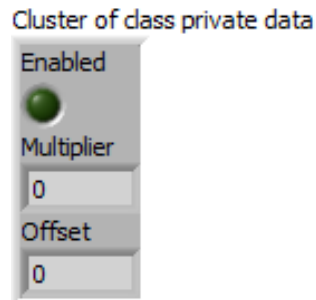
Alternative Configuration - Plug-in Parameters

About this task

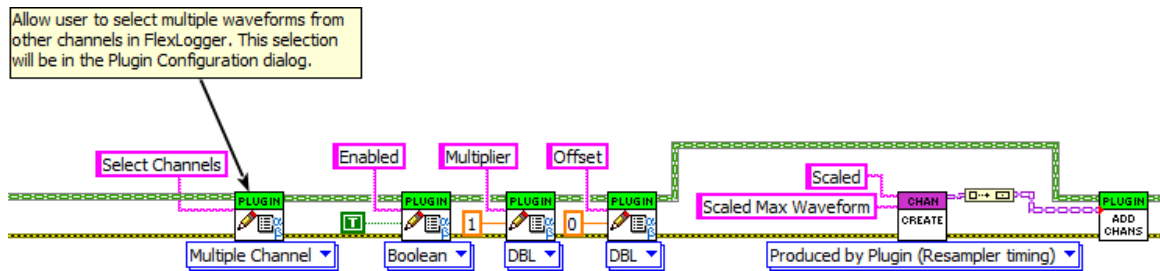
If you do not need to change settings while logging data, you can make Enabled, Multiplier, and Offset parameters of the plug-in instead of setpoint channels. Reconfigure the code from the previous example as follows:

Procedure

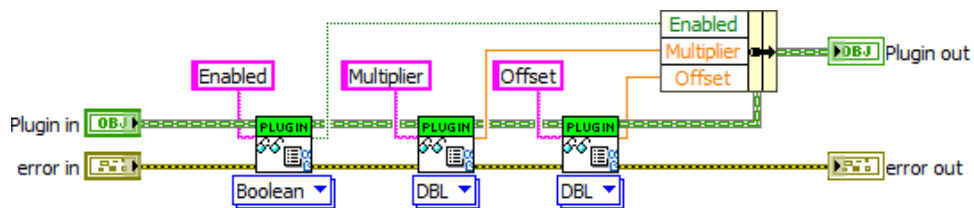
1. Update the class control to contain private data that can be passed between the VIs.



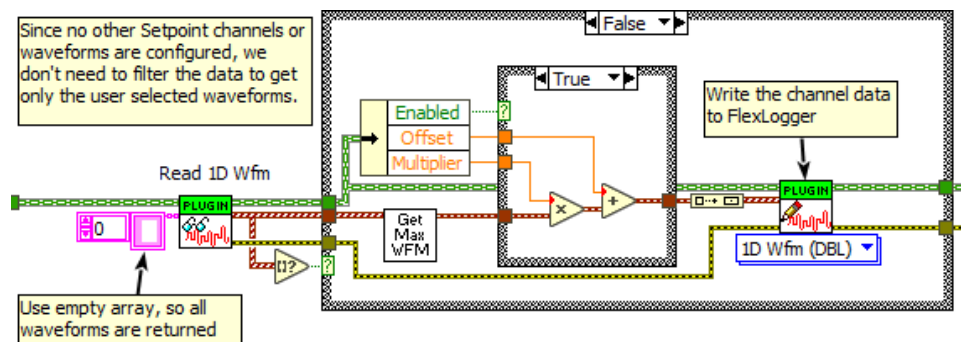
2. Modify Initialize VI to create the parameters for the plug-in.



3. Read the parameter values in Configure Session VI and update the private data. This VI is called whenever the user changes the parameter values from the **Plug-in Configuration** dialog box in FlexLogger.



4. Modify Process VI to use the Enabled, Multiplier, and Offset private data instead of parsing it out of the waveform data.



Below is the resulting plug-in configuration and channel table in FlexLogger:

Configure: New FlexLogger IO F

Name

New FlexLogger IO Plugin 1

Select Channels

Multiple sources mapped

Enabled

☒

Multiplier

10

Offset

200

OK

Cancel

	New FlexLogger IO Plugin 1 (cDAQ1Mod1/ai0, cDAQ1Mod1/ai1, cDAQ1Mod1/ai2)				
	CHANNEL NAME	LIVE VALUE	DETAILS		
Scaled	Scaled Max Waveform	242.634			

Alternative Configuration - Dynamic Channel Number with Individual Settings

About this task

You can configure the settings in FlexLogger to support a dynamic number of channels with separate settings. In this configuration, each channel has its own offset and multiplier setting as well as user-selected waveforms that each channel applies unique scaling parameters to. Reconfigure the code from the first example as follows:



NOTE
You can disable an entire channel in FlexLogger instead of keeping a separate channel parameter to disable it. To disable a channel, select the channel and click the **Disable** button on the right of the channel table.

	New FlexLogger IO Plugin 1					
	CHANNEL NAME	LIVE VALUE	DETAILS	ALARMS		
Scaled 1	Scaled Max Waveform 1	32.8087	Multiplier: 3, Offset: 22, Select Channels: cDAQ1Mod1/ai2			
Scaled 2	Scaled Max Waveform 2	Disabled	Multiplier: 1, Offset: 0, Select Channels: cDAQ1Mod1/ai7			Disable
Scaled 3	Scaled Max Waveform 3	265.078	Multiplier: 1, Offset: 0, Select Channels: cDAQ1Mod1/ai6, cDAQ1Mod1/ai7			

Procedure

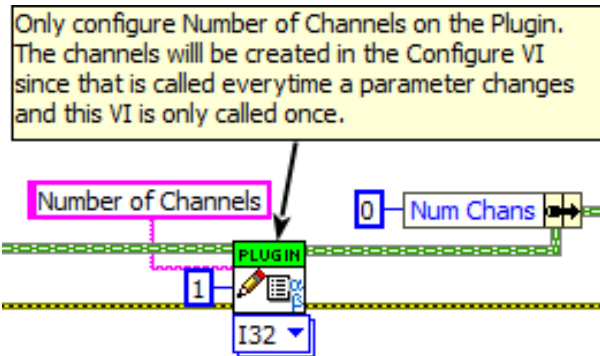
1. Update the class private data control to track the number of channels.

Cluster of class private data

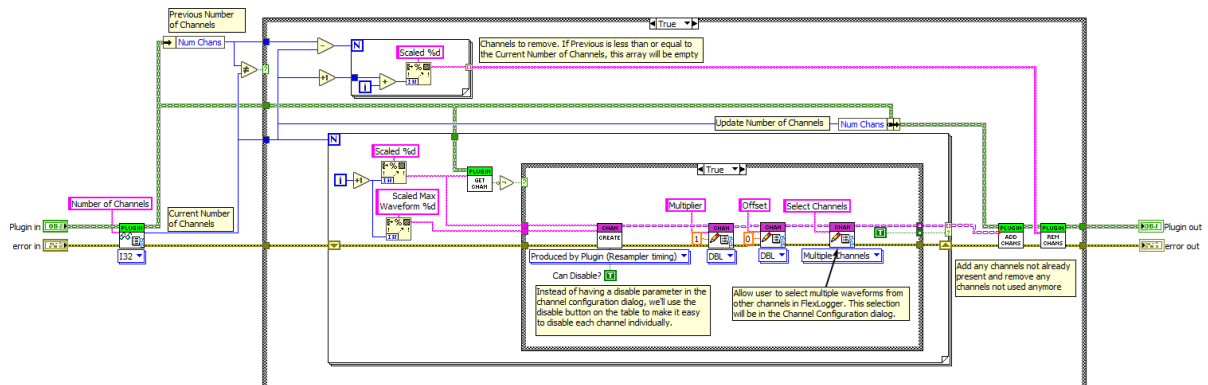
Num Chans

0

2. Modify Initialize VI to create the Number of Channels parameter for the plug-in and initialize the private data.



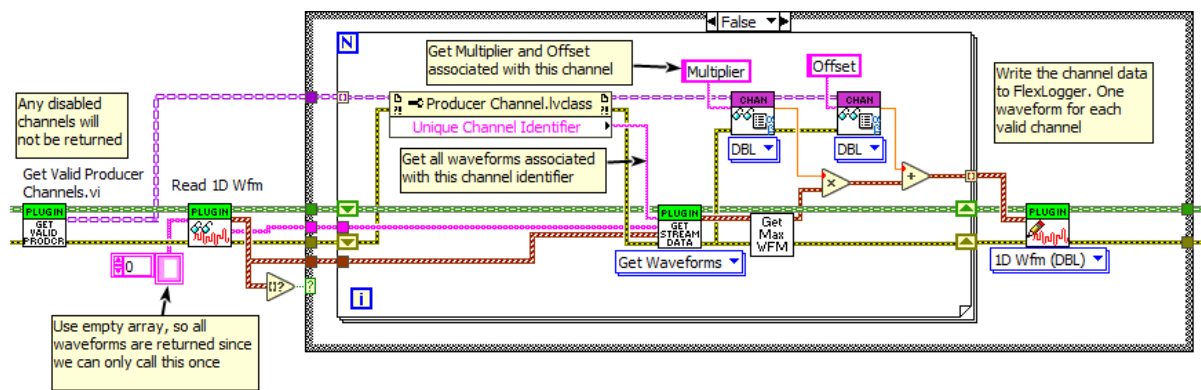
3. In the Configure Session VI, read the current number of channels configured by the user and compare with the previous value stored in the private data. Add any new channels that need to be configured and remove any channels that are no longer needed.



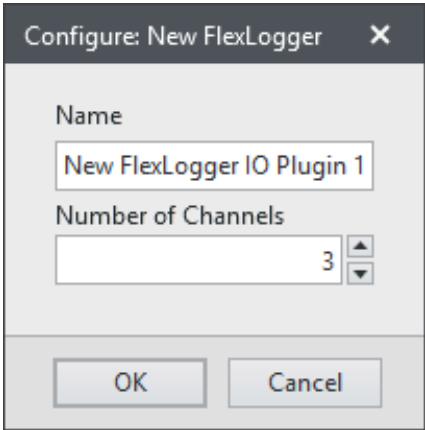
The **Get Channel With Name VI** (`<vi.lib>\FlexLogger\SDK\PluginSDK.lvlib\Plugin\`) determines if the channel has already been created. Since you allow the user to disable channels, disabled channels are not returned by the **Get Valid Channels VI**. You determine which channels to add and remove to prevent any existing channels from being removed or overwritten. This ensures all the configured settings for any existing channels remain unchanged, including the disabled setting. **Get Valid Channels VI** is called whenever the user changes the Number of Channels parameter from the **Plug-in Configuration** dialog in FlexLogger.

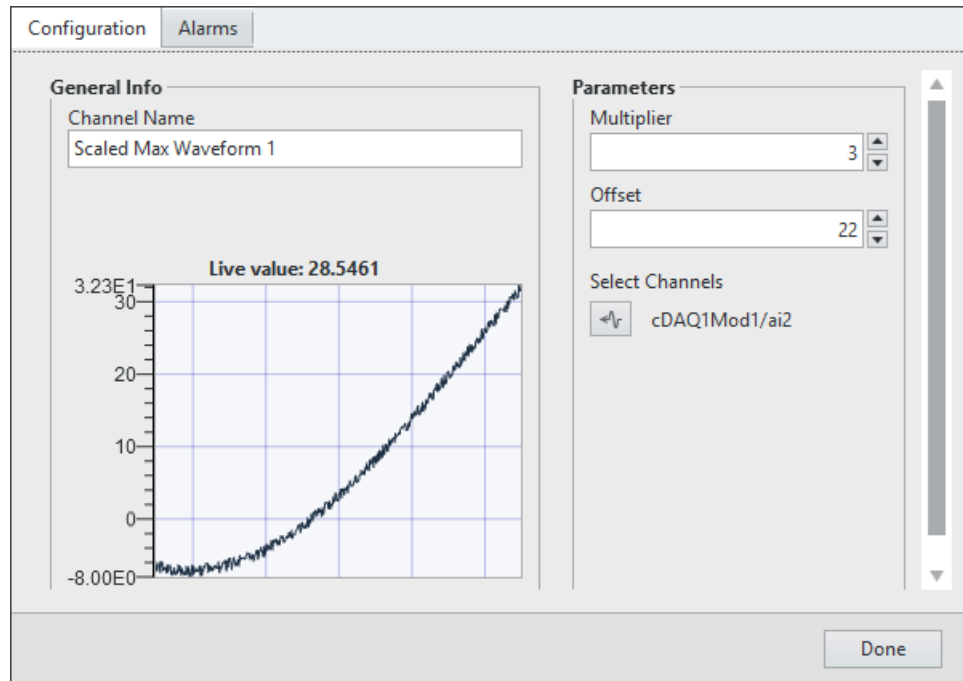
4. Modify the **Process VI** to go through all enabled channels. Use **Get Valid Producer Channels VI** (`<vi.lib>\FlexLogger\SDK\PluginSDK.lvlib\Plugin\`) to obtain only enabled channel producers for

this plug-in. For each channel, get the waveforms, multiplier, and offset associated with the channel.



Below is the resulting plug-in Configuration, Channel Configuration and channel table in FlexLogger:





	CHANNEL NAME	LIVE VALUE	DETAILS	ALARMS	
Scaled 1	Scaled Max Waveform 1	45.7509	Multiplier: 3, Offset: 22, Select Channels: cDAQ1Mod1/ai1,cDAQ1Mod1/ai2		
Scaled 2	Scaled Max Waveform 2	Disabled	Multiplier: 1, Offset: 0, Select Channels: cDAQ1Mod1/ai0		
Scaled 3	Scaled Max Waveform 3	8.89367	Multiplier: 1, Offset: 0, Select Channels: cDAQ1Mod2/ai0,cDAQ1Mod2/ai1		

Create Channel VI

The Create Channel VI allows you to create three types of channels, **Produced by Plug-in**, **Produced by Plugin (Resampler timing)**, and **Setpoint consumed by Plugin**. Creating channels involves the following information:

Table 9 : Plug-in Channel Components

Name	LabVIEW Data Type	Description
Unique Channel Identifier	string	Channel identifier that is unique in the plug-in context. Displayed in the far-left Channel Specification column (for example, ai0 for a DAQ channel). Strings with five or fewer characters display best in FlexLogger.
Default Channel Name	string	Channel name in FlexLogger.
Display Group	string	Name used in Channel Specification header and channel selector.
Unit	string	Channel units (for example, Hz).

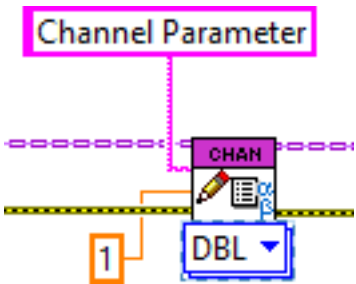
Table 9 : Plug-in Channel Components (Continued)

Name	LabVIEW Data Type	Description
Can Delete?	Boolean	Allow users to disable this channel. False by default. When setting Can Disable? to True, plug-in developers should monitor which channels are disabled by using Get Valid Channels . When a channel is disabled, do not write data back to FlexLogger for the disabled channel.
Default Value	double	Setpoint consumed by Plugin channels allow plug-in developers to set the default value that appears in FlexLogger when the plug-in is added.
dt(sec)	double	If Produced by Plugin channels use the 1D Wfm (DBL) or 1D Wfm (Digital) option provided by Write VI , this parameter tells the FlexLogger application what dt to expect.

FlexLogger allows users to change a channel’s **Default Channel Name** from both the channel configuration dialog and directly in the Channel Specification table. Additionally, FlexLogger requires all channels in the Channel Specification to have unique names—and will automatically change a plug-in channel name if that rule is violated (for example, myChannel becomes myChannel_1). Regardless of the change to **Default Channel Name**, the **Unique Channel Identifier** parameter remains unchanged.

Channel Specific Parameters

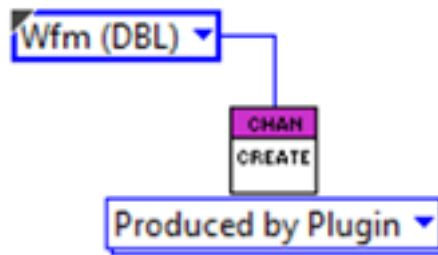
The Channel class supports double-precision floating point numbers, strings, enums, Booleans, and integers as channel-level parameters. Channel-specific parameters can be created by using the Channel class’s **Write Parameter VI**, as shown in the following example:



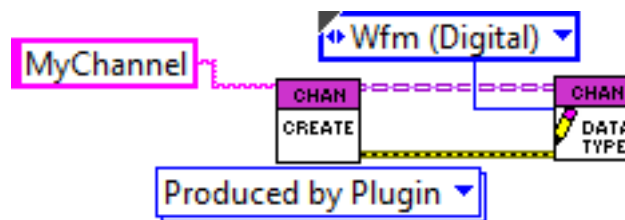
Channel Data Types

The Channel class supports analog (Wfm (DBL)), digital (Wfm (Digital)), and string data types for all channels. Analog waveform is the default data type.

To specify a data type, connect it to the **Create Channel VI**.



You can also set the data type using `Write Data Type VI` and connecting it to the `Create Channel VI`. The following example shows how to use `Write Data Type VI` to change the data type of a channel.

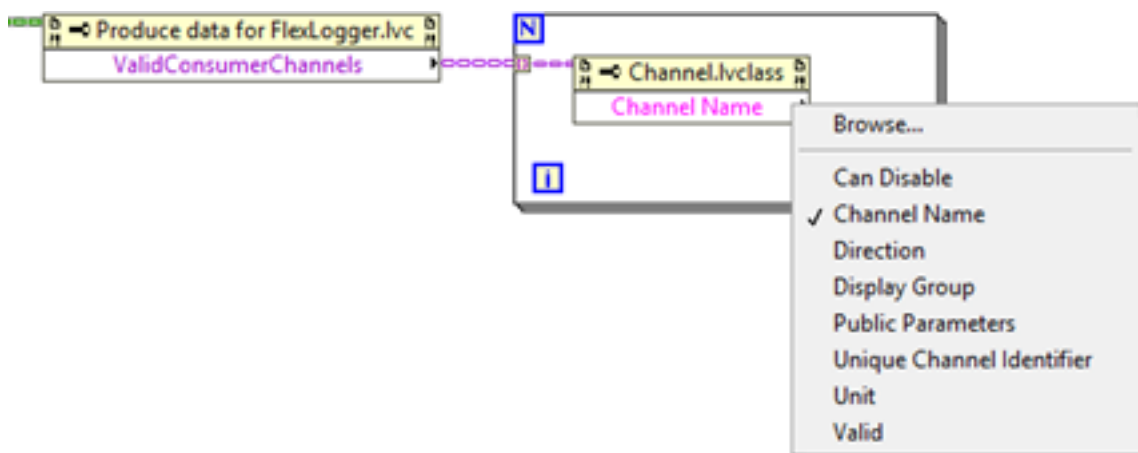


NOTE

FlexLogger cannot send string data to a plug-in.

Dynamic Channels

To help developers work with dynamic channels—such as when names change or channels are disabled—plug-ins maintain an array of valid channels. Wiring a property node to a plug-in class wire provides direct access to the `ValidProducerChannels`, and `ValidConsumerChannels` arrays:



Knowing what channels are valid allows developers to configure their plug-ins correctly and produce the right data for the available channels.

When channels assigned to a plug-in need to be changed, developers can use `Replace Producer Channels VI` and `Replace Channel Group VI` to replace multiple channels at once rather than individually. `Replace Producer Channels VI` replaces all

existing producer channels with a specified list of channels, which can be useful in situation where the user configuration changes in a way that requires a completely new set of producer channels. Replace Channel Group VI replaces all channels in a particular group with a specified list of channels, which can be useful in situations where only a subset of the channels need to be replaced.

System Interface Class

The System Interface class (`System Interface.lvclass`) defines the execution environment interacting with a FlexLogger plug-in. When FlexLogger loads and runs a built plug-in, the plug-in will communicate with FlexLogger’s underlying data engine. When debugging a plug-in in the Plug-in Environment Simulator, the plug-in will communicate with a simplified, test-specific environment.

Additional Classes

The following classes belong to `PluginSDK.lvlib` and are outside the scope of FlexLogger plug-in development. These classes should not be used in your plug-ins.

Table 10 : Additional Classes Not Used in Plug-in Development

Class Name	Summary
Addon Interface.lvclass	Defines the communication between plug-ins and FlexLogger
Message Completion Handler.lvclass	Messaging functionality for classes that do not inherit from <code>Plugin.lvclass</code>
Parameters.lvclass	Provides the parameter functionality for the Channel and Plug-in classes

FlexLogger Plug-in Debugging

Troubleshoot plug-in problems with the debugging resources available in LabVIEW and FlexLogger.

Determine which debugging resource best aligns with your troubleshooting scenario.

Table 11 : Plug-in Debugging Resources

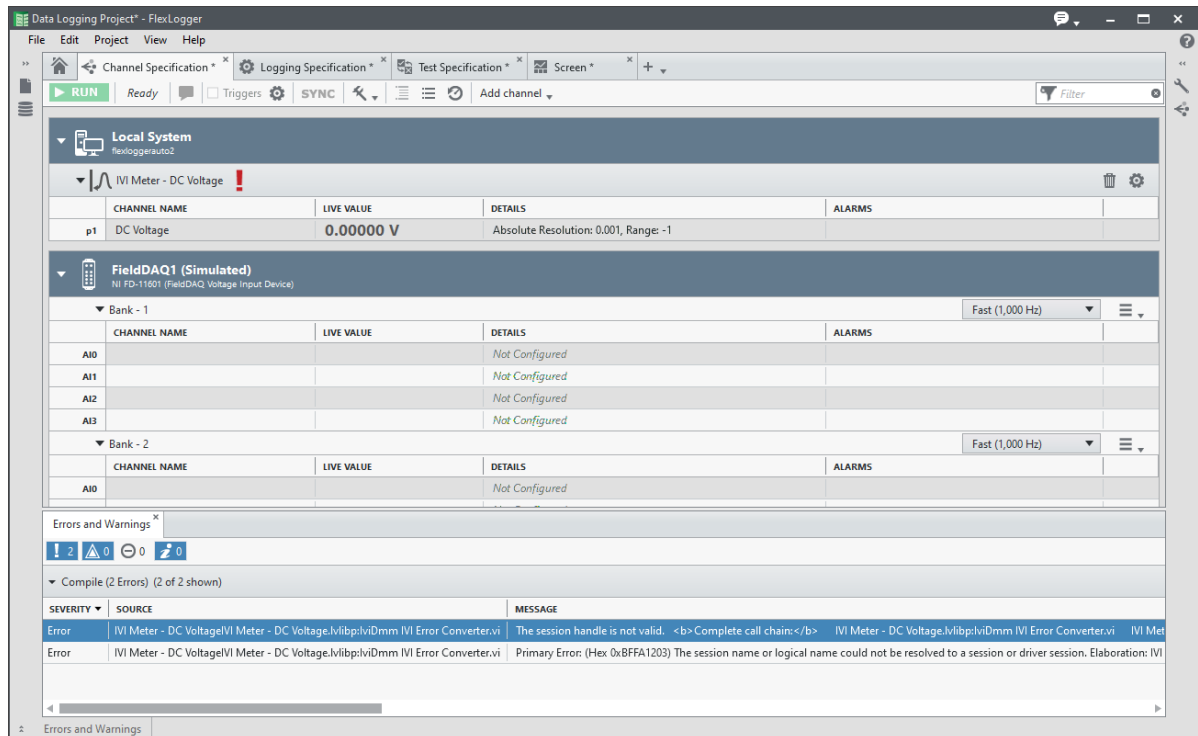
Scenario	Resource
I want to know why my plug-in is not available to load in FlexLogger.	Review the error log created by FlexLogger.
I want to view errors published by VIs I used in my plug-in in FlexLogger.	Configure your VIs in LabVIEW and view errors and warnings in FlexLogger.
I want to interactively test my plug-in before I use it in FlexLogger.	Use the Plug-in Environment Simulator in LabVIEW.

Debugging Plug-in Loading Errors

If FlexLogger fails to parse a plug-in XML file, the plug-in will not be available for users to add in the Channel Specification. In case of parsing failure, an `Error.txt` file will be created in the plug-in folder with details about the problem.

Viewing Errors in FlexLogger

Because the Plug-in class provides built-in error handling, it is important that developers wire their error wires to the provided error out terminals. For example, IVI class driver VIs publish errors on their error wires when they are initialized incorrectly. Wiring the error terminals from those driver VIs to the error out terminal ensures errors appear to users in FlexLogger:



Users receive three distinct error notifications:

- Detailed errors in the Errors and Warnings pane
- An error icon on the plug-in header
- The Error indicator by the **Stop Test** button

Interactively Testing Plug-ins

When plug-ins load and run in FlexLogger, the application's compiled data engine and the compiled plug-ins prevent interactive debugging with LabVIEW. The Plug-in Environment Simulator provides a simplified, test-specific environment that allows developers to run and debug plug-ins from the source G code.

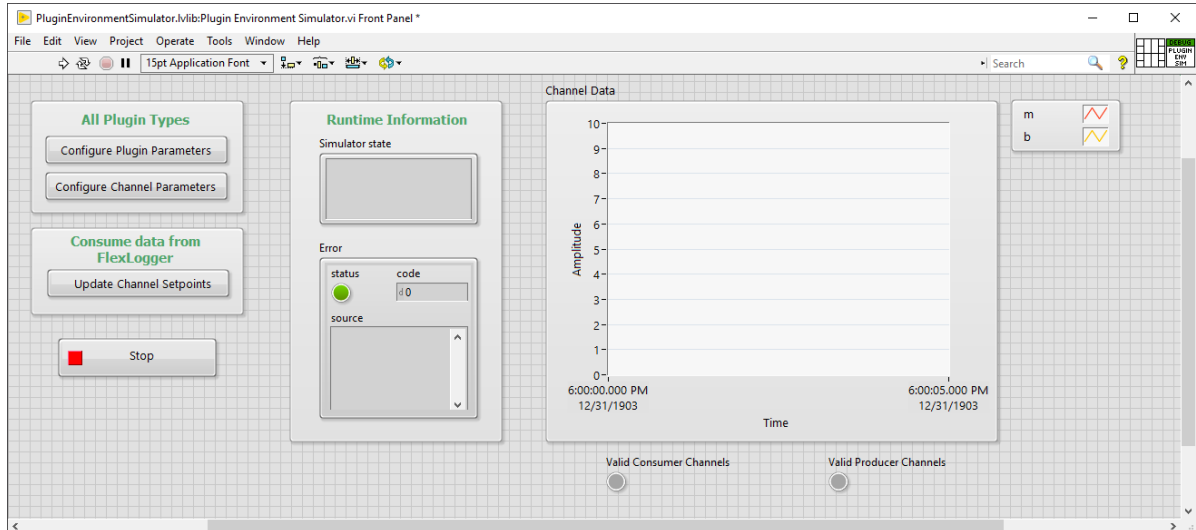
The Plug-in Environment Simulator installs to %ProgramFiles%\National Instruments\LabVIEW <version>\resource\FlexLogger\SDK\Plugin Environment Simulator.



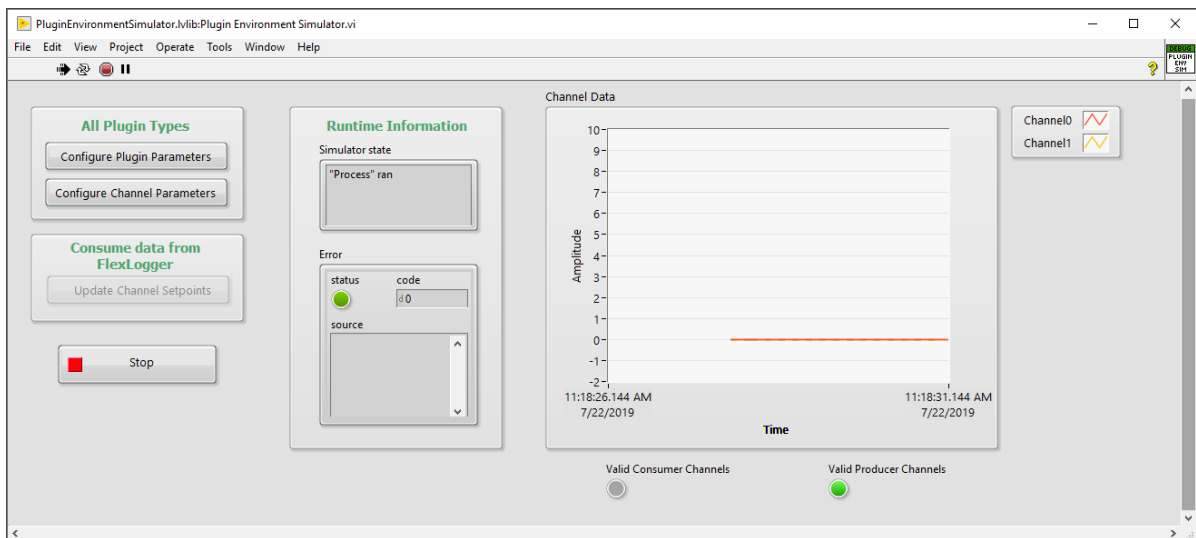
NOTE

The Plug-in Environment Simulator does not support digital channels.

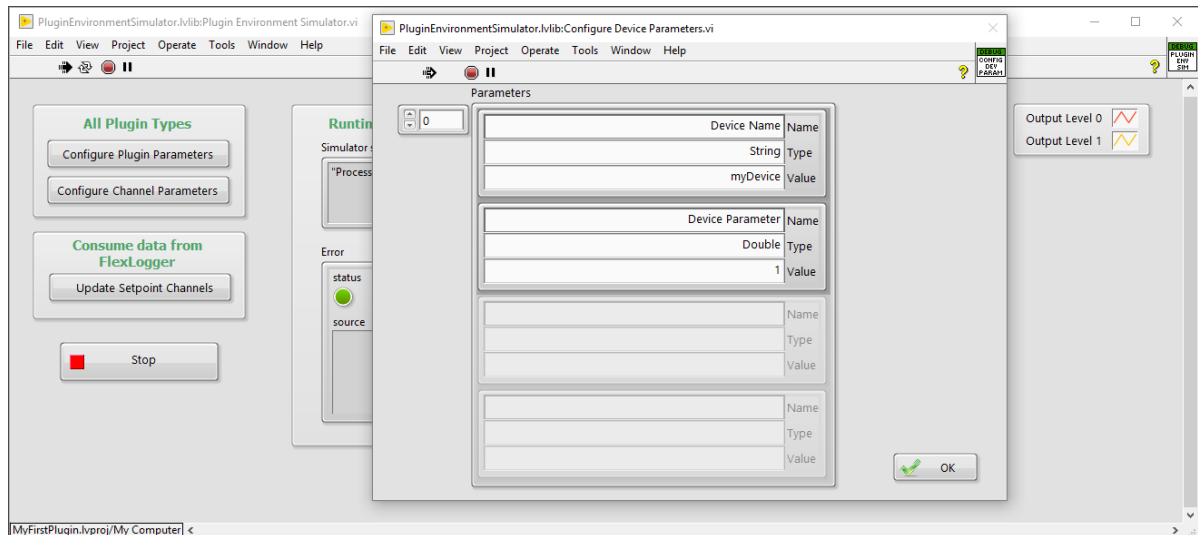
To enable interactive debugging of Plug-in source code, the Simulator is provided as a VI:



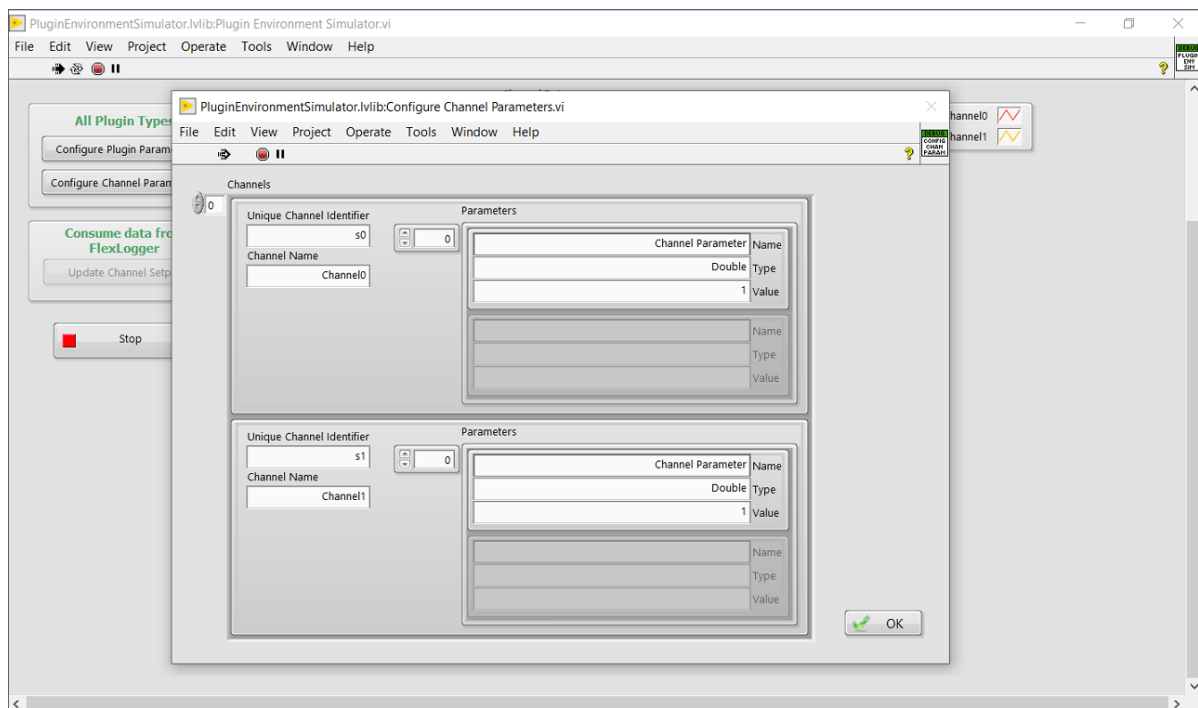
To debug a plug-in, open and run Plugin Environment Simulator VI. When the Simulator starts running, it prompts the user for the Plug-in class to debug. The following is the Simulator running an unmodified Produce data for FlexLogger plug-in:



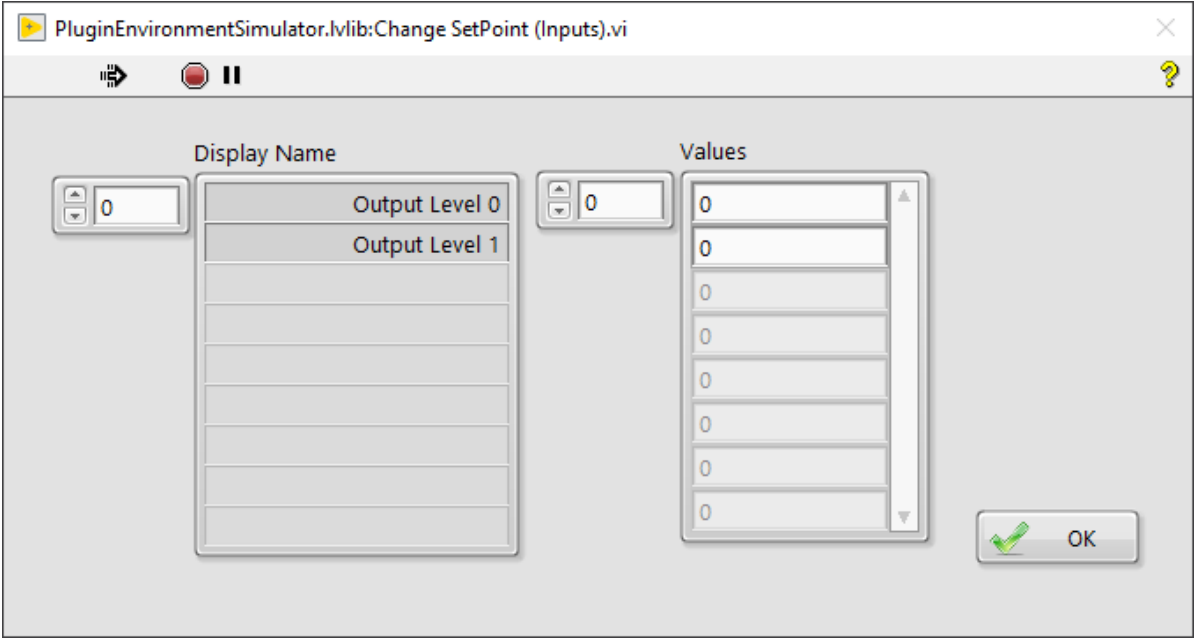
The **Configure Plugin Parameters** button allows users to update plug-in parameters:



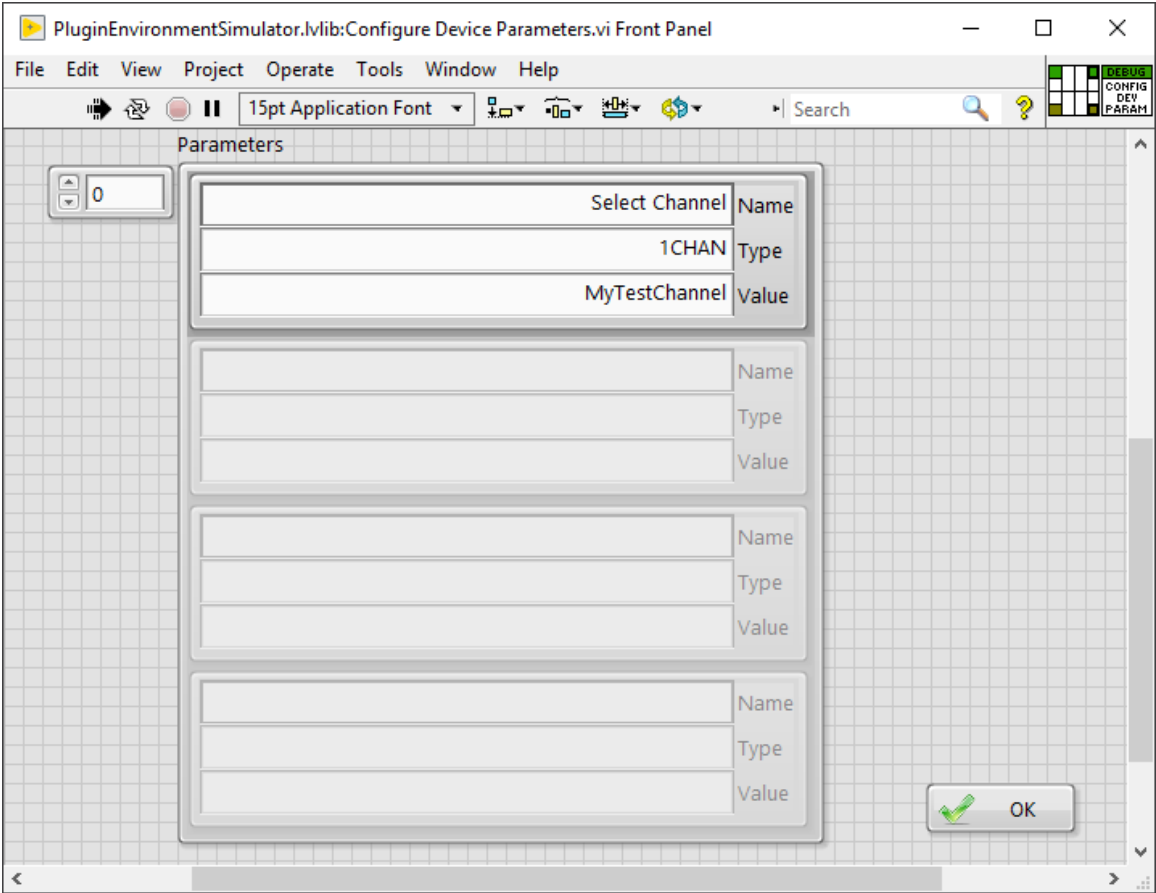
Similarly, the **Configure Channel Parameters** allows developer to update channel parameters:

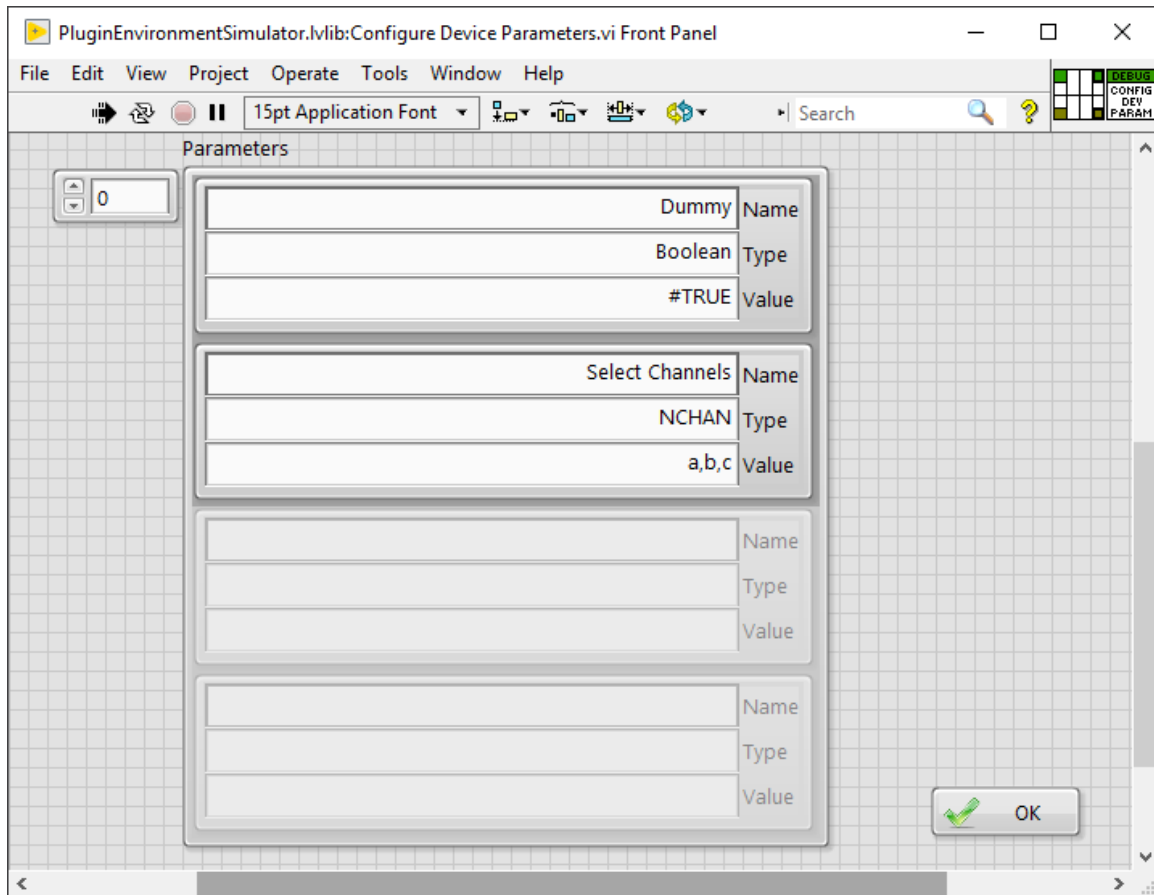


Plug-ins following the Consume setpoint data template do not write any data to FlexLogger, so the **Channel Data** chart shows the current setpoint values. The **Update Setpoint Channels** button provides the input information needed to write the correct values to an external system. The following shows setting the setpoint for an unmodified setpoint consumer plug-in example:

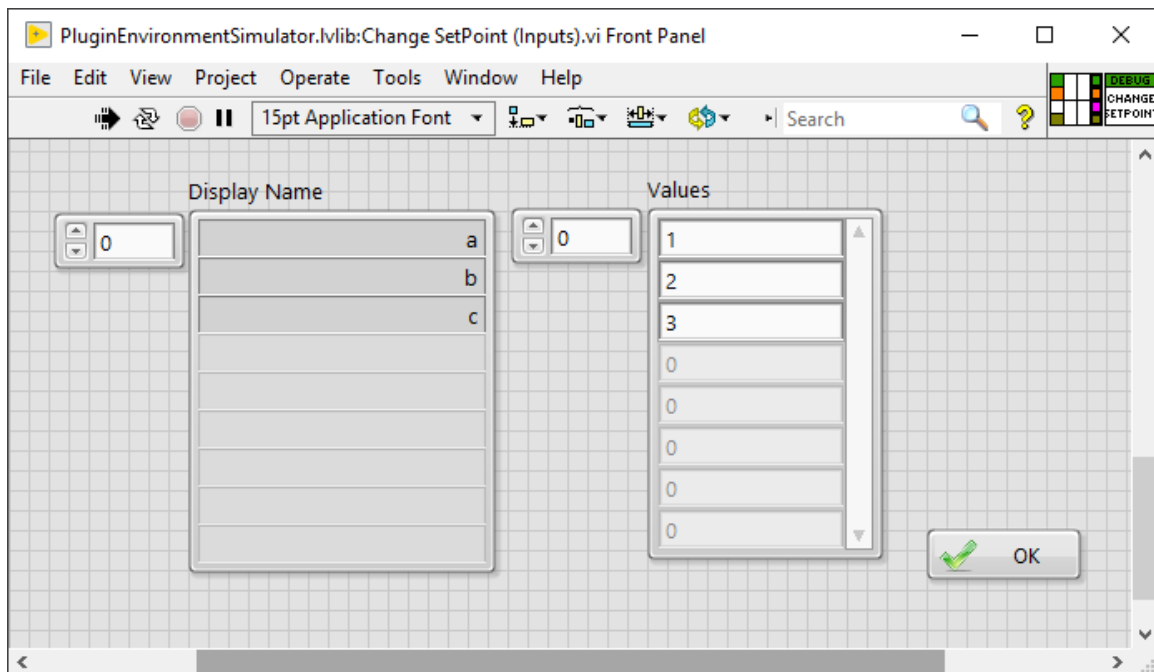


When using waveform channel inputs with the Plug-in Simulator, it is first necessary to pick channel(s) that are mapped to the parameters. This mimics the behavior when interacting with the application. In the Simulator, any designated name will receive a constant value waveform. Mapped channels are configured through the **Configure Device Parameters** dialog:





The constant value can be changed using the **Change Setpoint** option:



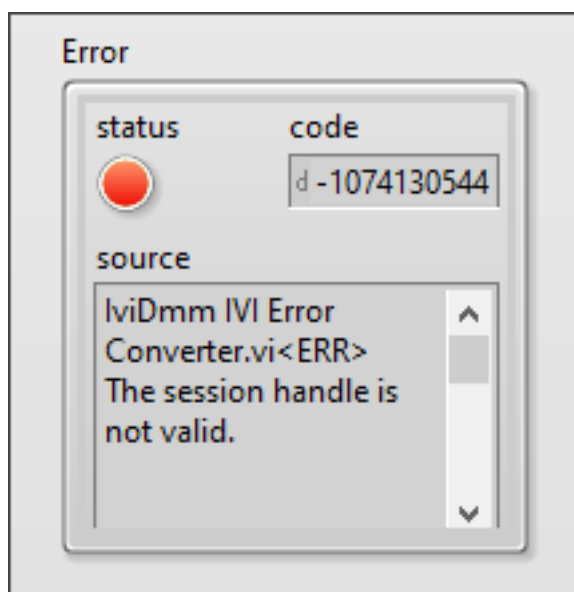
Interactive Debugging Session Example

About this task

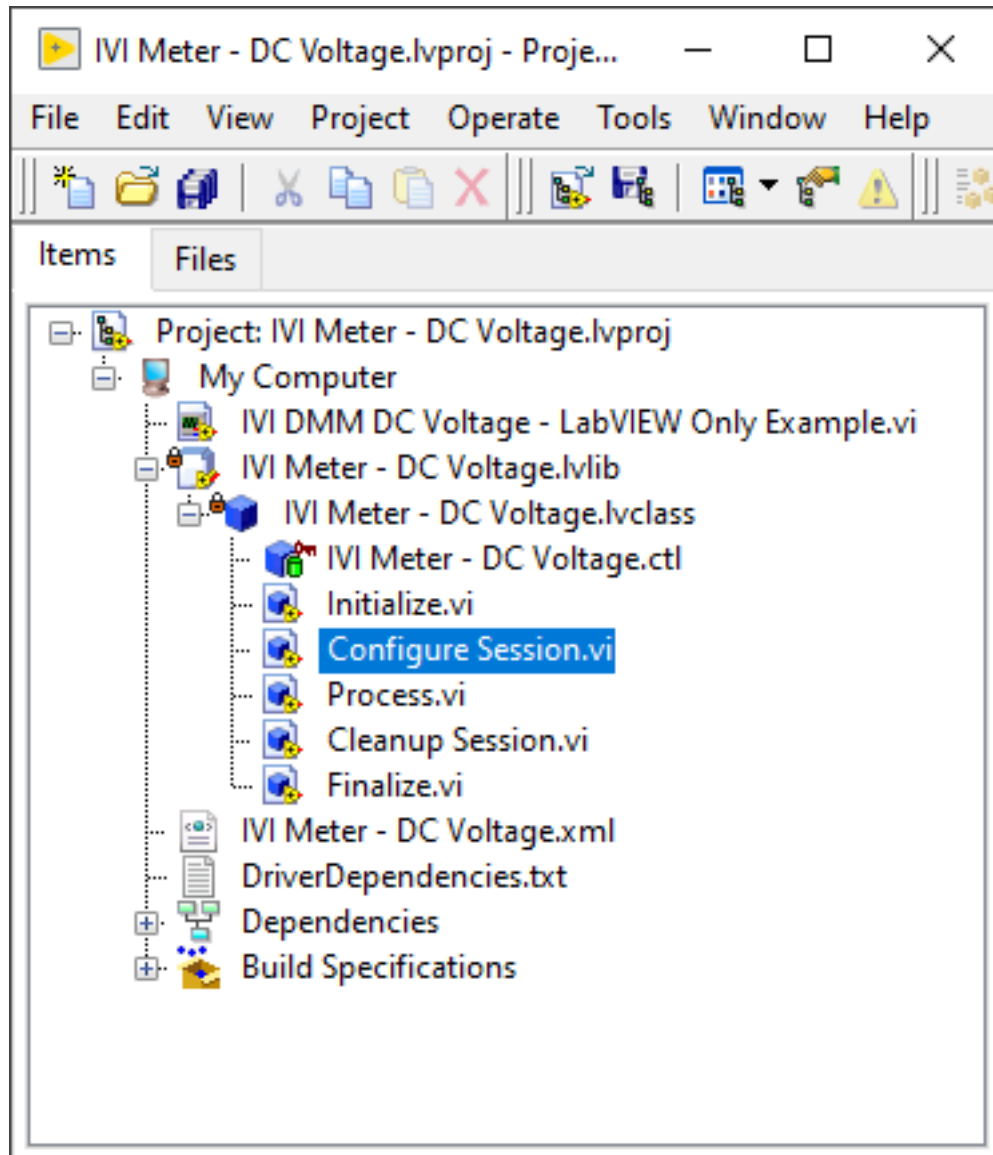
The following scenario illustrates how the Simulator could be used to investigate an issue with a plug-in under development.

Procedure

1. After loading and configuring the DMM plug-in, the plug-in channel does not produce any data. The **Error and Warning** pane in FlexLogger, displays an error message that references a problem in the `Configure Session VI`.
2. To investigate further, open the LabVIEW project for the DMM plug-in
`%ProgramFiles%\National Instruments\LabVIEW <version>\examples\FlexLogger\IO Plugins\IVI Meter - DC Voltage\IVI Meter - DC Voltage.lvproj`.
3. Open and run `Plug-in Environment Simulator VI` and then select the LabVIEW class of the DMM plug-in (`IVI Meter - DC Voltage.lvclass`).
4. After updating the plug-in parameter that defines the **IVI Logical Name** to test channel, the simulator stops running and the **Runtime Information** error terminal displays and error:



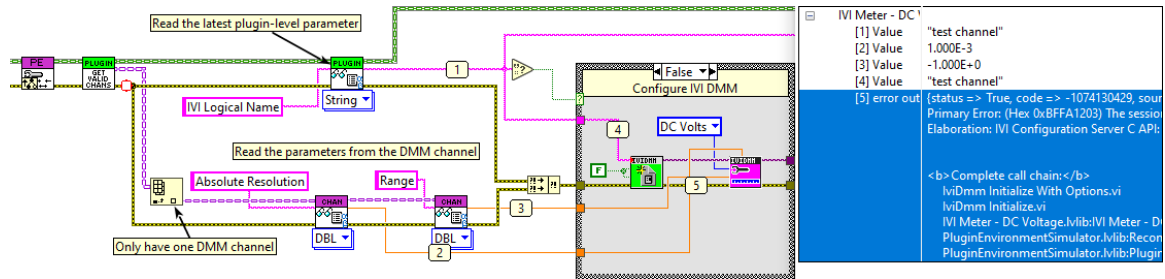
5. To investigate further, navigate to the plug-in LabVIEW project and open `Configure Session VI`.



6. With **Configure Session VI** open, add a breakpoint to see what part of the logic is generating the error:



7. With the breakpoint in place, run the simulator again and load the DMM plug-in. That action will hit the newly placed breakpoint. Using LabVIEW's debugging tools, step through the block diagram to see exactly which VI is having problems:



8. After this debugging session, you will see that the plug-in has been using the wrong **IVI Logical Name** parameter, resulting in an error from the IVI driver VIs. The plug-in runs as expected when using the correct name.

Information is subject to change without notice. Refer to the NI Trademarks and Logo Guidelines at ni.com/trademarks for information on NI trademarks. Other product and company names mentioned herein are trademarks or trade names of their respective companies. For patents covering NI products/technology, refer to the appropriate location: Help»Patents in your software, the patents.txt file on your media, or the National Instruments Patent Notice at ni.com/patents. You can find information about end-user license agreements (EULAs) and third-party legal notices in the readme file for your NI product. Refer to the Export Compliance Information at ni.com/legal/export-compliance for the NI global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data. NI MAKES NO EXPRESS OR IMPLIED WARRANTIES AS TO THE ACCURACY OF THE INFORMATION CONTAINED HEREIN AND SHALL NOT BE LIABLE FOR ANY ERRORS. U.S. Government Customers: The data contained in this manual was developed at private expense and is subject to the applicable limited rights and restricted data rights as set forth in FAR 52.227-14, DFAR 252.227-7014, and DFAR 252.227-7015.