

Technical Document

Niagara^{AX-3.x} NDIO Guide

February 13, 2009



Niagara^{AX} NDIO Guide

Copyright © 2009 Tridium, Inc.

All rights reserved.

3951 Westerre Pkwy, Suite 350

Richmond

Virginia

23233

U.S.A.

Copyright Notice

The software described herein is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Tridium, Inc.

The confidential information contained in this document is provided solely for use by Tridium employees, licensees, and system owners; and is not to be released to, or reproduced for, anyone else; neither is it to be used for reproduction of this Control System or any of its components.

All rights to revise designs described herein are reserved. While every effort has been made to assure the accuracy of this document, Tridium shall not be held responsible for damages, including consequential damages, arising from the application of the information contained herein. Information and specifications published here are current as of the date of this publication and are subject to change without notice.

The release and technology contained herein may be protected by one or more U.S. patents, foreign patents, or pending applications.

Trademark Notices

BACnet and ASHRAE are registered trademarks of American Society of Heating, Refrigerating and Air-Conditioning Engineers. Microsoft and Windows are registered trademarks, and Windows NT, Windows 2000, Windows XP Professional, and Internet Explorer are trademarks of Microsoft Corporation. Java and other Java-based names are trademarks of Sun Microsystems Inc. and refer to Sun's family of Java-branded technologies. Mozilla and Firefox are trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, Niagara^{AX} and Vykon are registered trademarks, and Workbench, WorkPlace^{AX}, and ^{AX}Supervisor, are trademarks of Tridium Inc. All other product names and services mentioned in this publication that is known to be trademarks, registered trademarks, or service marks are the property of their respective owners. The software described herein is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

CONTENTS

Preface	v
NDIO FAQs	v
Document Change Log	v
ndio Driver Installation	1-1
Ndio Quick Start	2-1
Configure the NdioNetwork	2-1
Add an NdioNetwork	2-1
<i>To add an NdioNetwork in the station</i>	2-1
Discover and add NdioBoards	2-1
<i>To discover and add NdioBoards</i>	2-1
Create Ndio proxy points	2-2
Using online Discover to add Ndio proxy points	2-2
<i>To discover I/O points</i>	2-2
<i>To add discovered I/O points as Ndio proxy points</i>	2-2
Manually adding Ndio proxy points	2-3
Niagara Ndio Concepts	3-1
About Ndio Architecture	3-1
About the Ndio Network	3-2
Ndio Network status notes	3-2
Ndio Network monitor notes	3-2
<i>About Ndio reboot logic</i>	3-3
Ndio Network tuning policy notes	3-4
Ndio Network poll scheduler notes	3-4
Ndio Network views	3-4
Ndio Board Manager	3-4
Ndio Board Manager usage notes	3-5
About the NdioBoard	3-5
NdioBoard properties	3-6
<i>Device status properties</i>	3-6
<i>Config and I/O-related properties</i>	3-6
Ndio Point Manager	3-7
Ndio Point Manager usage notes	3-7
<i>Add and Edit dialog fields</i>	3-7
<i>Universal Input selection notes</i>	3-8
<i>Output type selection notes</i>	3-9
About Ndio proxy points	3-9
Ndio point types	3-9
Ndio Proxy Ext common properties	3-10
Conversion types in Ndio proxy points	3-11
Scale and offset calculation (linear)	3-12
<i>Scale and offset calculation example 1</i>	3-12
<i>Scale and offset calculation example 2</i>	3-12

Non-linear sensor support 3-13
Non-linear sensor example 3-14
 Linear Calibration Ext 3-14
 BooleanInputPoint 3-15
 BooleanOutputWritable 3-16
 CounterInputPoint 3-16
About Ndio rate calculation 3-17
Properties 3-17
Actions 3-18
 ResistiveInputPoint 3-18
 ThermistorInputPoint 3-18
Type-3 Thermistor 3-19
Tabular Thermistor notes 3-19
Curve File Import/Export notes 3-20
 VoltageInputPoint 3-21
 VoltageOutputWritable 3-22

Ndio Plugin Guides..... 4-1

Ndio Plugin Guides Summary 4-1
ndio-NdioBoardManager 4-1
ndio-NdioPointManager 4-1

Ndio Component Guides..... 5-1

Ndio Component Reference Summary 5-1
ndio-FixedWindowRateType 5-1
ndio-LinearCalibrationExt 5-1
ndio-NdioBoard 5-1
ndio-NdioBoardFolder 5-1
ndio-NdioBooleanInputProxyExt 5-1
ndio-NdioBooleanOutputProxyExt 5-2
ndio-NdioCounterInputProxyExt 5-2
ndio-NdioHardwareOverride 5-2
ndio-NdioNetwork 5-2
ndio-NdioPointDeviceExt 5-2
ndio-NdioPointFolder 5-2
ndio-NdioPollScheduler 5-2
ndio-NdioResistiveInputProxyExt 5-2
ndio-NdioVoltageInputProxyExt 5-2
ndio-NdioVoltageOutputProxyExt 5-2
ndio-SlidingWindowRateType 5-2
ndio-ThermistorType3Type 5-2
ndio-TriggerRateType 5-2

PREFACE

Preface

- [NDIO FAQs](#)
- [Document Change Log](#)

NDIO FAQs

The following are frequently asked questions (FAQs) about the NiagaraAX NDIO driver:

Q: *What does NDIO (or Ndio) abbreviate?*

A: Niagara Direct Input Output, the original NiagaraAX driver for I/O hardware. Note throughout this document, capitalization of NDIO is typically limited to the leading “N”, as in “Ndio proxy point”. This reflects actual Niagara component and view names, such as NdioNetwork, NdioBoard, Ndio Board Manager, and so on. Occasionally, the lowercase “ndio” is used when referring to the actual ndio module (as in .jar file).

Q: *Does the newer NRIO (Nrio) driver replace the Ndio driver?*

A: No, the NRIO (Niagara Remote Input Output) driver supports a different group of I/O hardware. Although the two drivers (Ndio and Nrio) are very similar at the proxy point level, there are numerous differences at the component “network” and “device” levels.

Q: *Why is it when I add some Ndio proxy points, say a VoltageInputPoint, that they come up in fault?*

A: In the Ndio Point Manager if you add a VoltageInputPoint, ResistiveInputPoint, or ThermistorInputPoint and change the point’s Facets to another category of units, you typically see this fault status. This happens because those types of proxy points are automatically added with a “LinearCalibrationExt”, and its “units” property does not automatically stay in sync with the proxy point’s Facets.

To clear this fault status, go to the property sheet of the LinearCalibrationExt under the point, and assign the identical units as you did in the proxy point’s Facets. Alternatively, you could simply delete the point extension, but that would remove the “Offset” (calibration) utility that it provides.

Note this fault status happens most for VoltageInputPoints, and rarely (if ever) for a ThermistorInputPoint—because its units should always remain temperature, even if changing between °C and °F.

Document Change Log

Updates (changes/additions) to this *NiagaraAX NDIO Guide* document are listed below.

- Updated: February 13, 2009
Made various changes related to a new “Generic Tabular” conversion type available starting in 3.4.46 (note: requires `kitIo` module), including sections “[Conversion types in Ndio proxy points](#)” on page 3-11, “[Non-linear sensor support](#)” on page 3-13, and “[Curve File Import/Export notes](#)” on page 3-20. Added “[NDIO FAQs](#)” section in this Preface. In section on “[CounterInputPoint](#)” on page 3-16 reworded “Note” about count value maintained by I/O processor—count is maintained by station.
- Updated: December 18, 2008
Added new Ndio proxy points subsection “[Conversion types in Ndio proxy points](#)” on page 3-11, which includes the “Shunt500 Ohm Conversion” type added starting in AX-3.4 and in builds 3.3.28, 3.2.23, and 3.1.33. A bit more information was added about the different types, and related changes were made in sections “[Scale and offset calculation \(linear\)](#)” on page 3-12, and “[Linear Calibration Ext](#)” on page 3-14. In the latter section the “Fault Cause” property description was added, along with a Note about proxy point fault status related to the Linear Calibration Extension. A few other minor changes were made in the “[Niagara Ndio Concepts](#)” section.

- Updated: February 15, 2008
Converted document to “new look” print format. Changed document to reference the *NiagaraAX Drivers Guide*, a new document created from sections formerly in the *NiagaraAX User Guide*. Reversed the order of this change log to have latest entries first.
- Revised: August 24, 2005
Added new section “[Ndio Network monitor notes](#)” on page 3-2, explaining the [Ndio reboot logic](#) added to the Ndio monitor ping (see [Figure 3-3](#) on page 3). Moved copyright and trademark information to front of document and used new (PDF) cover design.
- Published: June 24, 2005
Added Copyright and Trademarks to preface.
- Draft: May 25, 2005
(Initial change log). Added an “[Ndio Quick Start](#)” section with simple procedures to [Configure the NdioNetwork](#) and [Create Ndio proxy points](#).
Made various minor changes in the “[Niagara Ndio Concepts](#)” section, including revised “[Ndio Network poll scheduler notes](#)” on page 3-4.

CHAPTER 1

ndio Driver Installation

To use the NiagaraAX ndio driver, you must have a target JACE host that provides either onboard I/O hardware points using Ndio (e.g. JACE-403), and/or that supports directly attachable I/O modules using Ndio (e.g. JACE-2/6). Ndio is not supported in Win32 JACE controllers, or certain other recent JACE platforms that use the *nrio* driver (Nrio, Niagara Remote Input Output) instead of Ndio.

From your PC, use the Niagara Workbench 3.*n.nn* installed with the “installation tool” option (checkbox “This instance of Workbench will be used as an installation tool”). This option installs the needed distribution files (*.dist* files) for commissioning various models of remote JACE platforms. The dist files are located under your Niagara install folder in a “sw\dist” subfolder.

Included in the dist file for each JACE platform is the necessary Ndio processor software, meaning underlying files and firmware used by a host JACE to communicate with either onboard (or attached) Ndio boards. Ndio support is automatically installed when you use the “Distribution File Installer” in a platform session with the JACE, or the distribution file option in the platform’s “Commissioning Wizard.” Note that the dist file includes many *other* things besides just Ndio support.

Apart from installing the 3.*n.nn* version of the Niagara distribution file in the JACE, make sure to install the **ndio** module (if not already present, or upgrade if an older revision). If using AX-3.4 or later, be sure to install the **kitIo** module too. For more details, see “[About the Commissioning Wizard](#)” in the *JACE NiagaraAX Install and Startup Guide*.

Note: The **kitIo** module provides a “Generic Tabular” conversion selection for non-linear sensor support, plus has several types of thermistor “curve files” in its *xml1* folder. AX-3.4 or later is required in the JACE.

Following this, the remote JACE is now ready for Ndio software configuration in its running station, as described in the rest of this document. See “[About Ndio Architecture](#)” on page 3-1.

Note: Basic procedures for using the Ndio driver, including online discovery of I/O points (and their addition to your Niagara station), is in the next section. See “[Ndio Quick Start](#)” on page 2-1.

CHAPTER 2

Ndio Quick Start

This section provides a collection of procedures to use the Ndio driver to make Ndio proxy points, the station interface to physical I/O points. As with other NiagaraAX drivers, you can do most configuration from special “manager” views and property sheets using Workbench.

These are the main subsections:

- [Configure the NdioNetwork](#)
- [Create Ndio proxy points](#)

Configure the NdioNetwork

To add and configure the NdioNetwork, perform the following main tasks:

- [Add an NdioNetwork](#)
- [Discover and add NdioBoards](#)

Add an NdioNetwork

To add an NdioNetwork in the station

Use the following procedure to add an [NdioNetwork](#) under the station’s Drivers container.

To add an NdioNetwork in the station:

- Step 1 Double-click the station’s Drivers container, to bring up the **Driver Manager**.
- Step 2 Click the **New** button to bring up the New DeviceNetwork dialog. For more details, see “[Driver Manager New and Edit](#)” in the *Drivers Guide*.
- Step 3 Select “NdioNetwork,” number to add: 1, and click **OK**.
This brings up a dialog to name the network.
- Step 4 Click **OK** to add the NdioNetwork to the station.
You should have an NdioNetwork named “NdioNetwork” (or whatever you named it), under your Drivers folder.

Discover and add NdioBoards


To discover and add NdioBoards


In Ndio architecture, NdioBoards act as “device-level” components. An NdioBoard represents one or more Ndio *processors*, servicing some number of I/O points (see “[About the NdioBoard](#)” on page 3-5).

Depending on JACE hardware platform, the number of NdioBoards to be discovered varies:

- A JACE, with integral (onboard) I/O has only a *single* NdioBoard.
- A JACE-2/6, with one or more attached I/O modules, has a separate NdioBoard for each module. Other JACE models may have an externally-attached I/O module, which appears as an NdioBoard.

To discover NdioBoards:

- Step 1 Double-click the **NdioNetwork**, or:
right-click the NdioNetwork and select **Views > Ndio Board Manager**.
This brings up the **Ndio Board Manager**.
- Step 2 Click the **Discover** button  to launch an Ndio Board Discovery job.
A progress bar appears at the top of the view, and updates as the discovery occurs.

- Step 3 When the discovery job completes, discovered **NdioBoards** are listed in the *top pane* of the view, in the “Discovered” table (Figure 3-4). The bottom pane, labeled “Database,” is a table of NdioBoards that are currently mapped into the Niagara station—initially, this table will be empty.
- Step 4 Click to *select all* discovered NdioBoards, then click the Add button . The **Add** dialog appears, in which you can accept all defaults (see Figure 3-5 on page 5).
- Step 5 Click **OK** to add the NdioBoard(s) to your station.
See the next section: “Create Ndio proxy points”.

Create Ndio proxy points

As with device objects in other drivers, each **NdioBoard** has a Points extension that serves as the container for proxy points. The default view for any Points extension is the Point Manager (and in this case, the **Ndio Point Manager**). You use it to create Ndio proxy points under any NdioBoard.

This section provides quick start procedures for both tasks, as follows:

- [Using online Discover to add Ndio proxy points](#)
- [Manually adding Ndio proxy points](#)

Note: For general information, see “About the Point Manager” in the Drivers Guide.

Using online Discover to add Ndio proxy points

This is the recommended way to accurately add Ndio proxy points under an NdioBoard. Use the following procedures:



- [To discover I/O points](#)
- [To add discovered I/O points as Ndio proxy points](#)

Note: If your NdioNetwork has multiple NdioBoards, repeat both procedures (*discover and add*) for each NdioBoard, until you have all I/O points proxied in the station.

To discover I/O points

Perform this task to discover I/O points.

To discover I/O points of an NdioBoard:

- Step 1 In the **Ndio Device Manager**, in the **Exts** column, double-click the **Points** icon  in the row representing the NdioBoard you wish to explore.
This brings up its **Ndio Point Manager**.
- Step 2 Click the **Discover** button  to learn what I/O points are on the NdioBoard.
When the discovery job completes, discovered I/O points are listed in the *top pane* of the view, in the “Discovered” table. Each I/O point occupies one row.
- Step 3 To add Ndio proxy points, see “[To add discovered I/O points as Ndio proxy points](#)”.

To add discovered I/O points as Ndio proxy points

Perform this task to add Ndio proxy points.

To add discovered I/O points as Ndio proxy points:

- Step 1 Select the I/O point or points in the Database pane of the **Ndio Point Manager**.
- Step 2 You can map selected points in the station in different ways:
 - Drag from the Discovered pane to Database pane (brings up an **Add** dialog).
 - Double-click an item in the Discovered pane (also brings up an **Add** dialog).This works the same as in other driver’s Point Manager views.
- Step 3 When the **Add** dialog appears, you typically edit a number of fields for each I/O point. For complete details, see “[Ndio Point Manager usage notes](#)” on page 3-7.
The following brief summaries explain Add dialog fields:
 - **Name** is the Ndio proxy point name—you typically change this to describe the I/O purpose.
 - **Type** is the Ndio “type,” which is selectable for any “universal input,” but fixed for any output. See “[Universal Input selection notes](#)” on page 3-8 and “[Output type selection notes](#)” on page 3-9.
Note: Unlike other entries in the Add dialog, you cannot edit the point’s Type later.
 - **Address** is the learned I/O point’s hardware address.
 - **Poll Frequency** specifies the point’s poll frequency group (default is Normal).
 - **Conversion** specifies the conversion type used between the Ndio proxy extension’s “Device Fac-

- ets” and the parent point’s facets. See “[Conversion types in Ndio proxy points](#)” on page 3-11.
- **Facets** are the Ndio proxy point’s facets, for how the value should be displayed in Niagara.
- Step 4 When you have Ndio proxy point(s) configured properly for your usage, click **OK**.
The proxy points are added to the station, and appear listed in the Database pane.
For more details, see “[About Ndio proxy points](#)” on page 3-9.

Note: *In cases where you notice a fault status for input points you have edited with “non-default” facets, this is likely related to a mismatch of the “Units” in the point’s automatically-appended **Linear Calibration Ext**. To clear this fault status, expand the point’s Linear Calibration Ext and edit Units to match the units in the point’s facets.*

For more information, see “[Linear Calibration Ext](#)” on page 3-14.

Manually adding Ndio proxy points

You can manually add Ndio proxy points, using the New button in the Ndio Point Manager, or by dragging from the ndio palette. However, this method is generally not recommended.

CHAPTER 3

Niagara Ndio Concepts

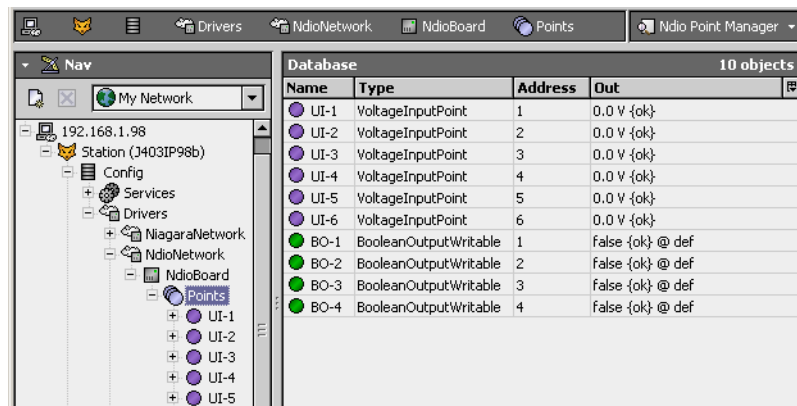
This section provides conceptual details on the Ndio driver (Niagara direct input/output) and its components, including views. Ndio components are the station interface to physical I/O points on certain JACE controllers (including JACE-2/6 platform) and/or directly-attachable I/O expansion modules.

About Ndio Architecture

Essentially, Ndio uses the standard NiagaraAX network architecture. See “[About Network architecture](#)” in the *Drivers Guide* for more details. For example, real-time data is modeled using Ndio proxy points, which reside under an NdioBoard “device”, which in turn resides under an NdioNetwork container in the station’s DriverContainer (Drivers).

Hierarchically, the component architecture is: network, board, points extension, points ([Figure 3-1](#)).

Figure 3-1 Ndio driver architecture



Name	Type	Address	Out
UI-1	VoltageInputPoint	1	0.0 V {ok}
UI-2	VoltageInputPoint	2	0.0 V {ok}
UI-3	VoltageInputPoint	3	0.0 V {ok}
UI-4	VoltageInputPoint	4	0.0 V {ok}
UI-5	VoltageInputPoint	5	0.0 V {ok}
UI-6	VoltageInputPoint	6	0.0 V {ok}
BO-1	BooleanOutputWritable	1	false {ok} @ def
BO-2	BooleanOutputWritable	2	false {ok} @ def
BO-3	BooleanOutputWritable	3	false {ok} @ def
BO-4	BooleanOutputWritable	4	false {ok} @ def

Unlike most other drivers, Points is the *only extension* under an NdioBoard “device,” and the sole purpose of the Ndio driver—to configure (and proxy) the actual Ndio I/O hardware points.

Note: You use “Manager” views of Ndio container components to add all Ndio components to your station, including Ndio proxy points. In these views, the Ndio driver provides online “discovery” of available hardware (Learn Mode), which greatly simplifies engineering.

For more details see:

- [About the Ndio Network](#)
- [Ndio Board Manager](#)
- [About the NdioBoard](#)
- [Ndio Point Manager](#)
- [About Ndio proxy points](#)

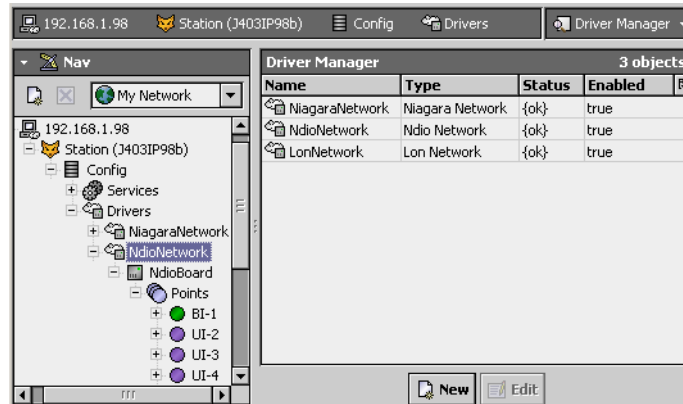
About the Ndio Network

The NdioNetwork is the top-level container component for “everything Ndio” in a station.

Note: *Only one NdioNetwork component is valid in a station—regardless of how many onboard I/O points and/or attached I/O boards are present.*

The NdioNetwork should reside in the station’s DriverContainer (“Drivers”). The simplest way to add an NdioNetwork is from the “Driver Manager” view, using the “New” command. Or, you can simply copy the NdioNetwork from the ndio palette into Drivers.

Figure 3-2 NdioNetwork in DriverContainer



Note: *The JACE platform must have the ndio module installed. Otherwise, an error occurs explaining that the ndio module is missing. If this occurs, install the **ndio** module in that JACE controller and repeat the operation.*

The NdioNetwork component has the typical collection of slots and properties as most other network components. For details, See “Common network components” in the *Drivers Guide*. The following sections explain further:

- [Ndio Network status notes](#)
- [Ndio Network monitor notes](#)
- [Ndio Network tuning policy notes](#)
- [Ndio Network poll scheduler notes](#)
- [Ndio Network views](#)

Ndio Network status notes

Unlike most “fieldbus” drivers such as Bacnet and Lonworks, the status of an NdioNetwork can be “down,” in addition to the normal “ok” or less typical “fault” (fault might result from misconfiguration of a network component). A down status might occur in the case of external I/O where the physical connection to the JACE was removed (no communications are possible).

The Health slot contains historical timestamp properties that record the last network status transitions from ok to any other status. The “Fault Cause” property further explains any fault status.

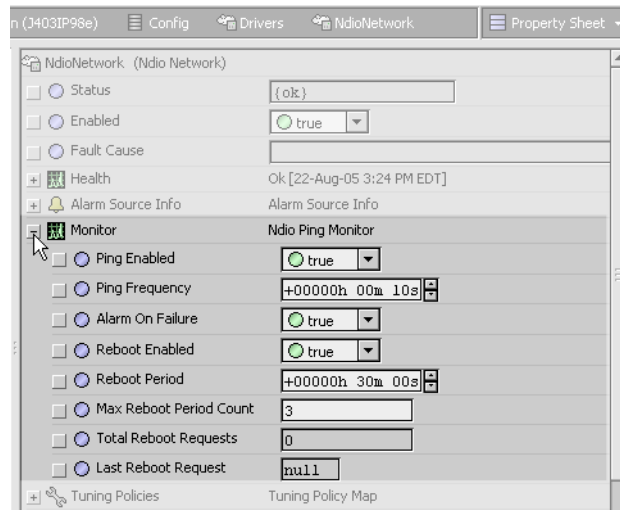
Note: *As in other driver networks, the NdioNetwork has an available “Alarm Source Info” container slot you can use to differentiate Ndio network alarms from other component alarms in the station. See “About network Alarm Source Info” in the Drivers Guide for more details.*

Ndio Network monitor notes

The network’s monitor routine verifies child NdioBoard component(s)—the “pingable” device equivalent in Ndio. For general information, see “About Monitor” in the *Drivers Guide*.

The Ndio ping monitor provides *added functionality* that allows the Ndio driver to *reboot* the JACE if serious problems are detected with I/O functionality. This [Ndio reboot logic](#) is configurable in several properties under the **Monitor** slot of the NdioNetwork, as shown in [Figure 3-3](#).

Figure 3-3 Monitor properties of NdioNetwork



The reboot-related Monitor properties include the following:

- **Reboot Enabled**
If set to true (default), and the NdioNetwork is enabled, the JACE reboots whenever [Ndio reboot logic](#) detects a serious issue. (If NdioNetwork is not running, only the Ndio processor is reset).
- **Reboot Period**
Default is 30 minutes. Defines the amount of time in which the maximum number of reboots can occur. If more than the configured number of reboots occurs within this period, the network's Enabled property is set to false, and the JACE is rebooted.
- **Max Reboot Period Count**
Default is 3. Specifies the maximum number of reboots that can occur within the reboot period (above) before Ndio is disabled.
- **Total Reboot Request**
(Read-only status) Specifies how many times [Ndio reboot logic](#) has rebooted the JACE.
- **Last Reboot Request**
(Read-only status) Provides a timestamp of when [Ndio reboot logic](#) last rebooted the JACE.

About Ndio reboot logic

As of build 3.0.88, the Ndio driver includes added “reboot logic” to automatically correct conditions where the JACE or Ndio components stop communicating effectively. This logic allows the Ndio processor(s) to behave in a predictable fashion in the case of a problem.

The following scenarios are addressed:

1. Station stops communicating with Ndio daemon.
By design, once the station starts communicating with the Ndio daemon, it must continue to communicate. If communication fails or is halted, the Ndio daemon stops talking to the Ndio processor, and the Ndio processor is reset. If the NdioNetwork is still running, this is detected, and the JACE is rebooted. If the NdioNetwork is not running, the Ndio processor resets itself and stays in that state until the next reboot cycle.
2. Ndio daemon stops communicating with the Ndio processor.
If the Ndio daemon fails to communicate with the Ndio process for *any* reason after talking to it at least once, the Ndio processor will reset. If an NdioNetwork is in the running station, this reset may result in a JACE reboot.
3. The Ndio processor stops responding to requests from the Ndio daemon.
If the Ndio processor resets or stops responding, the results in an error condition reported to the NdioNetwork, which may choose to reboot the JACE.

Note: Whenever any Ndio-requested reboot is about to occur, the station is automatically saved, and messages are sent to the event log prior to rebooting. Additionally, an alert is generated.

Under the NdioNetwork's Monitor slot, several properties are available to configure this Ndio reboot logic (see [Figure 3-3](#) on page 3).

Ndio Network tuning policy notes

The NdioNetwork has the typical network-level Tuning Policy Map slot with a single default Tuning Policy, as described in “About Tuning Policies” in the *Drivers Guide*.

However, given the small number of total Ndio proxy points supported under an NdioNetwork, the importance of creating additional tuning policies (or editing the default tuning policy) is generally lower relative to other driver networks.

Ndio Network poll scheduler notes

The NdioNetwork has the typical network-level Poll Scheduler slot with standard collection of properties, as described in “About poll components” in the *Drivers Guide*. The following notes apply to the Poll Scheduler of an Ndio network:

- By default, discovered/added Ndio proxy points have a Poll Frequency setting of “Normal” at the default Normal Rate of 1 second. Given the small number of total Ndio proxy points supported under an NdioNetwork, you can safely assign them to the “Fast” rate (default 0.333 seconds).
- The fastest useful poll rate is 200ms, the fastest Ndio can update its values over the I2C-bus. However, in the case of a JACE with maximum number of external I/O modules attached, it is somewhat slower than that (as more Ndio processors are serviced).

Ndio Network views

The NdioNetwork’s default view is the **Ndio Board Manager**, equivalent to the Device Manager in most other drivers. You use this view to discover and add NdioBoard components to the station. For details, see “Ndio Board Manager” on page 3-4.

Other standard views are also available on the NdioNetwork. However, apart from the Ndio Board Manager, you typically access only its property sheet.

Ndio Board Manager

As the default NdioNetwork view, the Ndio Board Manager has an online “Learn Mode” to find I/O modules attached (and/or integral) to the host JACE (Figure 3-4). You use this view to discover and add one or more “device-level” NdioBoard components to the station database (Figure 3-5). For general information, refer to “About the Device Manager” in the *Drivers Guide*.

Figure 3-4 Adding NdioBoard discovered using Ndio Board Manager

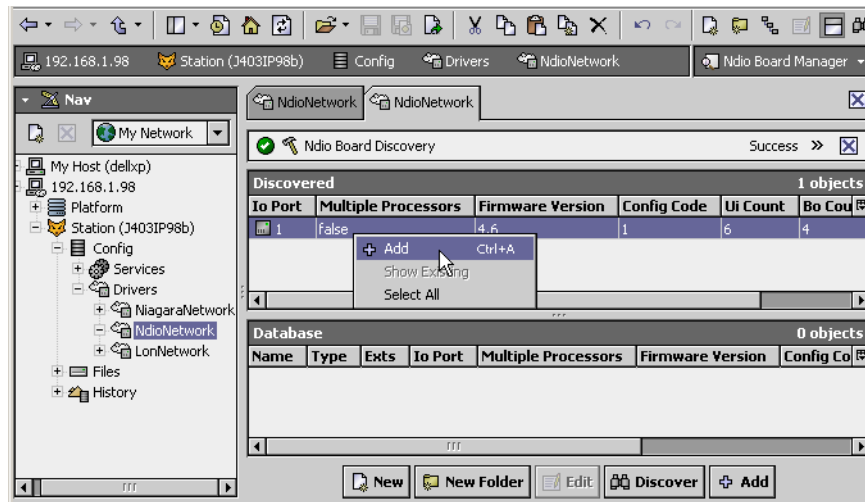
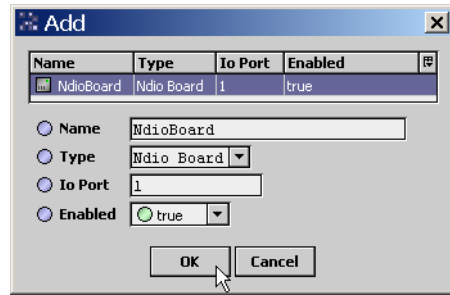


Figure 3-5 Add dialog for NdioBoard(s)



After adding all discovered NdioBoard(s), you use the [Ndio Point Manager](#) view of each NdioBoard to add Ndio proxy points—one for each available I/O hardware point (terminal address).

- [Ndio Board Manager usage notes](#)

Ndio Board Manager usage notes

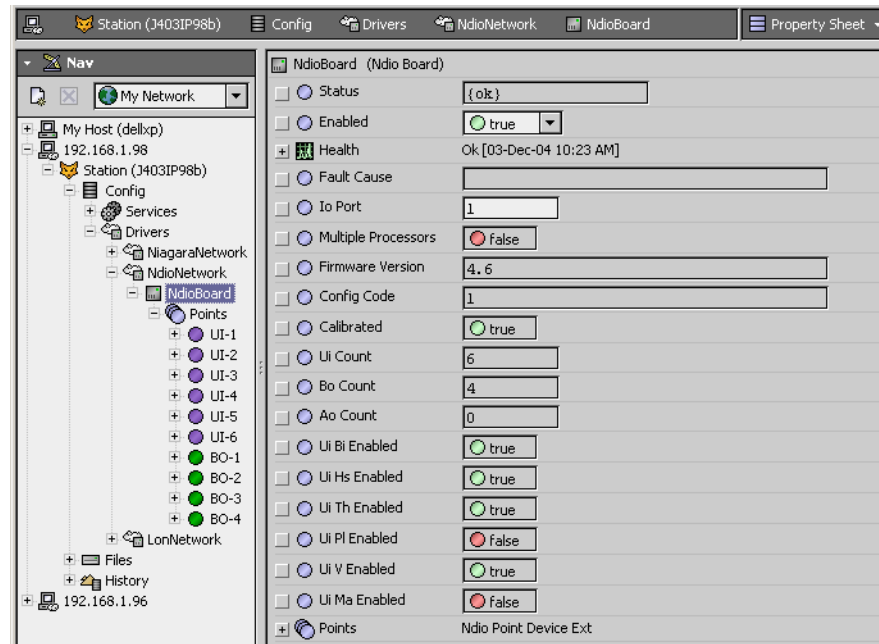
Before using discover, ensure that any external I/O modules are correctly installed and cabled to the host JACE controller. These other notes also apply:

- Offline engineering is possible (using New), but not recommended. There is no match feature.
- In general, the “New Folder” feature has little practical application.
- Usually you do not change any defaults in the Add dialog (Figure 3-5) when adding an NdioBoard, nor use the Edit dialog (to edit same properties) after adding one.

About the NdioBoard

The NdioBoard represents a board with Ndio *processors* that service hardware I/O points. The host JACE controller communicates with Ndio processors using the “device” model of NdioBoard (parent) to physical I/O points serviced by its processors as (child) Ndio proxy points.

Figure 3-6 NdioBoard property sheet



A JACE with integral (onboard) I/O, e.g JACE-403, is represented by a *single* NdioBoard. If a JACE has directly attached I/O module(s), each physical module is represented by a separate NdioBoard. Note that some I/O modules may have multiple (more than 1) Ndio processors, such as an IO-32 module. An NdioBoard component reflects this by status property “Multiple Processors,” (either false or true).

Under an NdioNetwork, each NdioBoard must have a unique “Io Port” number, starting from 1. Apart from the “Enabled” property, this is the *only* writable property of an NdioBoard. All other [NdioBoard properties](#) are read-only types ([Figure 3-6](#)), reflecting the status and I/O capabilities of the Ndio hardware.

Note: *The [Ndio Board Manager](#) view of the NdioNetwork provides online “Learn Mode” discover and add commands to ensure all necessary NdioBoards are created and configured properly for the host’s hardware configuration. See “[Configure the NdioNetwork](#)” on page 2-1.*

Each NdioBoard has a single important device extension: Points—the container for all its Ndio proxy points. Each proxy point represents a specific I/O terminal address, serviced by that board’s I/O processor(s). For more details, see “[Ndio Point Manager](#)” on page 3-7.

NdioBoard properties

NdioBoard properties can be categorized into two groups:

- [Device status properties](#)
- [Config and I/O-related properties](#)

Note: *As in other driver networks, the NdioBoard has an available “Alarm Source Info” container slot you can use to differentiate NdioBoard alarms from other component alarms in the station. See “[Device Alarm Source Info](#)” in the Drivers Guide for more details.*

Device status properties

NdioBoard has typical device-level status properties (see “[Device status properties](#)” in the *Drivers Guide*). The following notes apply:

- **Status**
Status of NdioNetwork communications to this NdioBoard. Possible status flags include:
 - ok - Normal communications, no other status flags.
 - disabled - Enabled property is set to false, either directly or in NdioNetwork. While status is disabled, all child Ndio points have disabled status; NdioBoard polling is suspended.
 - fault - Typically configuration error, e.g. NdioBoard (duplicate or invalid Io Port number).
 - down - Error communicating to one or more Ndio processors on Ndio board.
- **Enabled**
Either true (default) or false. Can be set directly or in parent NdioNetwork. See Status disabled description above.
- **Health**
Contains properties including timestamps of last “ok” time and last “fail” time, plus a string property describing last fail cause.
- **Fault Cause**
If status has fault, describes the cause (e.g.: “invalid ioPort provided (1 <= ioPort <= 4)”).

Config and I/O-related properties

In addition to common device-level [Device status properties](#), NdioBoard has the following unique properties relating to port communications and I/O capabilities. Apart from Io Port (set automatically if NdioBoard was discovered and added), all properties are read-only types.

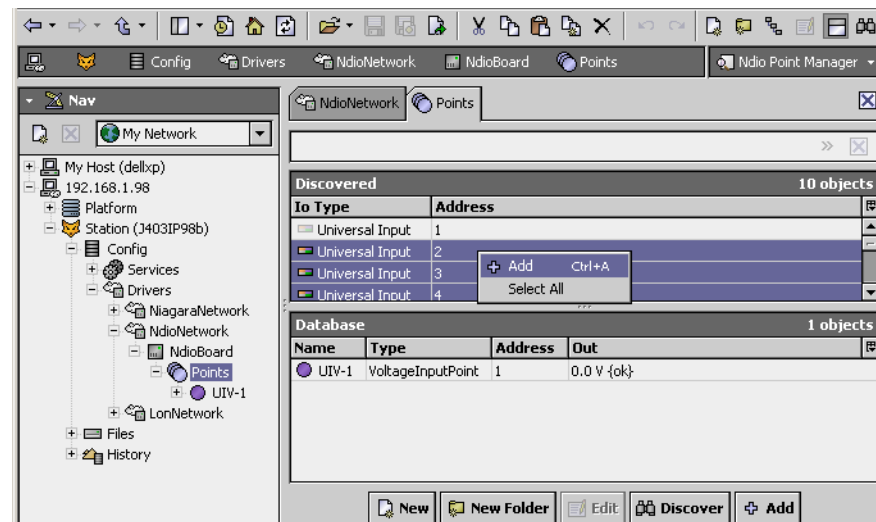
- **Io Port**
Integer between 1 and 4, unique among all NdioBoards under the NdioNetwork.
- **Multiple Processors**
Either false or true. For most NdioBoard components, will be false (such as for JACE-403 onboard I/O, or for IO-20 module). Is true only if Ndio board has multiple Ndio processors.
- **Firmware Version**
Reflects firmware revision of the board’s Ndio processor(s).
- **Config Code**
A config code numeral corresponding to some known complement of Ndio hardware.
- **Calibrated**
Either false or true. Shows if Ndio hardware is factory-calibrated. Should always be true.
- **Ui Count**
Number of available I/O universal inputs (UIs) on this board (*may exist as proxy points*).
- **Bo Count**
Number of available I/O boolean outputs (BOs) on this board (*may exist as proxy points*).
- **Ao Count**
Number of available I/O analog outputs (AOs) on this board (*may exist as proxy points*).

- **Ui Bi Enabled**
Either false or true. Shows if boolean inputs (BI) is supported by the board’s universal inputs.
 - **Ui Hs Enabled**
Either false or true. Shows if high-speed counter inputs are supported by the board’s UIs.
 - **Ui Th Enabled**
Either false or true. Shows if resistive (thermistor) inputs are supported by the board’s UIs.
 - **Ui PI Enabled**
Either false or true. Shows if resistive (platinum) inputs are supported by the board’s UIs.
 - **Ui V Enabled**
Either false or true. Shows if a DC voltage input is supported by the board’s UIs.
 - **Ui Ma Enabled**
Either false or true. Shows if a DC milliamp (mA) input is supported by the board’s UIs.
- Note: Most UIs support a 0-to-20mA input with an addition of a 499 ohm resistor and using an Ndio proxy point for voltage input.*

Ndio Point Manager

As the default view for the Points extension under an [NdioBoard](#), the Ndio Point Manager provides an online “Learn Mode” to find available I/O terminal points ([Figure 3-7](#)). You use this view to discover and add corresponding Ndio proxy points to the station database.

Figure 3-7 Adding Ndio proxy points discovered using Ndio Point Manager



Typically you add a proxy point for *each* discovered I/O terminal. For more details, see “[Ndio Board Manager usage notes](#)” on page 3-5. For procedures, see “[Create Ndio proxy points](#)” on page 2-2.

For general information, see “[About the Point Manager](#)” in the *Drivers Guide*.

Ndio Point Manager usage notes

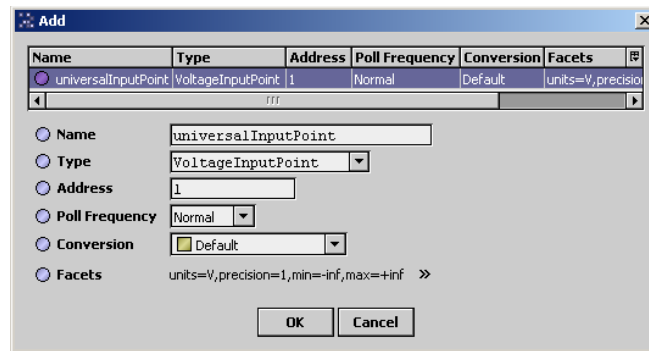
The following notes apply when using the Ndio Point Manager:

- [Add and Edit dialog fields](#)
- [Universal Input selection notes](#)
- [Output type selection notes](#)

Add and Edit dialog fields

When adding Ndio proxy points for discovered I/O points, the following items are available in the Add dialog ([Figure 3-8](#))

Figure 3-8 Add dialog in Ndio Point Manager



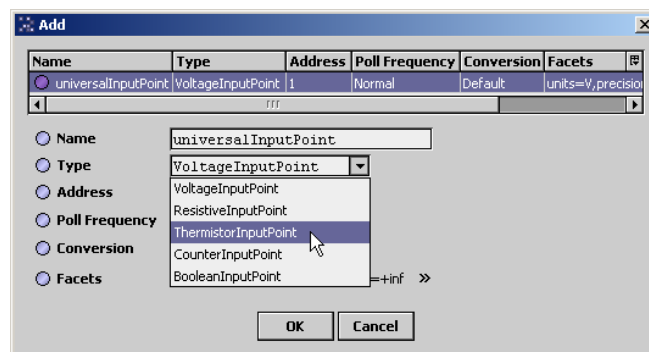
- **Name**
 Name of the Ndio proxy point (equivalent to right-click Rename, can be edited anytime). Typically, you edit name from the default name to reflect the actual purpose of the I/O point, such as “Room101T,” “AHU-1_FanStatus,” and so forth.
Note: The number appended onto a default name has no association to the I/O point address.
- **Type**
 The type of Ndio proxy point to create (*not* editable after adding, see “Universal Input selection notes” on page 3-8.)
- **Address**
 The I/O terminal address within that general IO type. Recommended to be left at default.
- **Poll Frequency**
 The assigned poll frequency group, as configured in the NdioNetwork’s poll scheduler. See “Ndio Network poll scheduler notes” on page 3-4.
- **Conversion**
 The conversion type used between units in the proxy extension’s “Device Facets” and the units in the parent point’s Facets. See “Ndio Proxy Ext common properties” on page 3-10 for more details.
- **Facets**
 Facets for the Ndio proxy point. Equivalent to accessing facets through the point’s property sheet (and can be edited anytime). For more details, see “About point facets” in the *Drivers Guide*.

Universal Input selection notes

All Ndio-capable JACE controllers and external I/O modules provide some number of “universal input” (UI) terminals. For example, the JACE-403 provides 6 UI inputs (terminals UI1—UI6). Unlike some other I/O boards where you must “hardware configure” each UI-type input using a jumper or switch, you do UI terminal configuration in *software*, from the Ndio Point Manager.

Do this at *add time* by selecting the needed input type in the Add dialog, as shown in Figure 3-9.

Figure 3-9 Select Ndio input type for each universal input



For most I/O platforms, any of the following are valid universal input choices:

- **VoltageInputPoint**
 NumericPoint that reads a 0-to-10Vdc input signal and produces either a voltage value or linear scaled output value.
- **ResistiveInputPoint**
 NumericPoint that reads a resistive signal within a 0-to-100K ohm range and produces either a ohms value or linear scaled output value.

- [ThermistorInputPoint](#)
NumericPoint that reads a Thermistor temperature sensor (Type 3, or other) signal and produces a scaled output value.
- [CounterInputPoint](#)
NumericPoint that counts the number of contact closures at the input, and also calculates a rate value. One of these two values is configurable as the status numeric output value. Rate type is configurable as either fixed window, sliding window, or trigger type.
- [BooleanInputPoint](#)
BooleanPoint that reads the current input as one of two boolean states (equipment status).

Any selection but BooleanInputPoint results in a standard NumericPoint, but with a different type Ndio input proxy extension. The BooleanInputPoint results in a BooleanPoint with an NdioBooleanInput proxy extension. For more details, see “About Ndio proxy points” on page 3-9.

Note: After adding any Ndio proxy point, you can edit name, address, conversion, and facets if desired—but not type. To change type, you must delete the point and then add it again, selecting from the Type drop-down menu, as shown in Figure 3-9. The one exception here is the [ResistiveInputPoint](#) and [ThermistorInputPoint](#), which are actually the same—except for the conversion type used in the ProxyExt.

Output type selection notes

Ndio-capable JACE controllers and external I/O modules typically have some number of digital output (DO, either relay or triac type) and/or 0-to-10Vdc analog output (AO) terminals. For any discovered output I/O terminal, the Add dialog preselects the appropriate *writable* Ndio point type, as either:

- [BooleanOutputWritable](#)
A standard BooleanWritable point, but with a NdioBooleanOutput proxy extension.
- [VoltageOutputWritable](#)
A standard NumericWritable point, but with NdioVoltageOutput proxy extension.

Unlike when adding universal input points, there is no alternate selection of type available when adding output points.

About Ndio proxy points

Ndio proxy points are similar to other driver’s proxy points. You can (and often do) add alarm and history extensions to them, and link them into other station logic as needed. For general information, see “About proxy points” in the *Drivers Guide*.

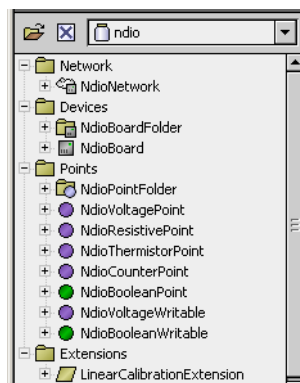
The following sections explain further:

- [Ndio point types](#)
- [Ndio Proxy Ext common properties](#)
- [Conversion types in Ndio proxy points](#)
- [Scale and offset calculation \(linear\)](#)
- [Non-linear sensor support \(AX-3.4 and later\)](#)
- [Linear Calibration Ext](#)

Ndio point types

As seen in the **ndio** palette (Figure 3-10), there are 7 different Ndio point types.

Figure 3-10 Ndio components as seen in the ndio palette



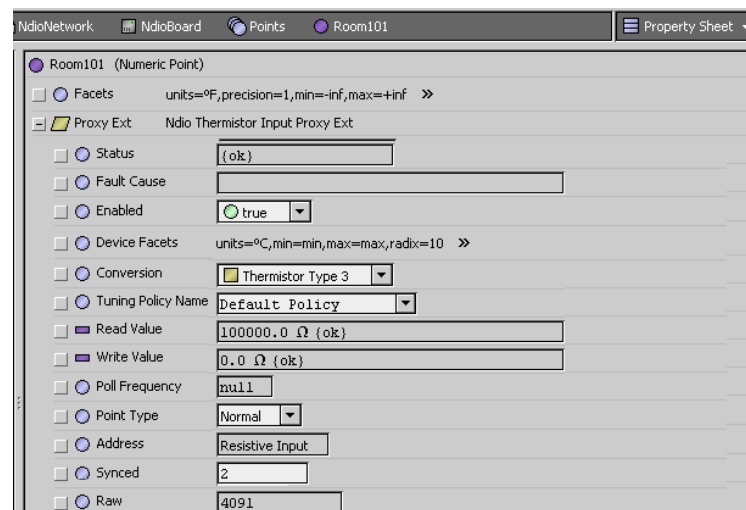
These 7 different Ndio point types derive from the following 4 control point types:

- **BooleanPoint**
For boolean “universal input” type NdioBooleanPoint ([BooleanInputPoint](#)).
- **NumericPoint**
For numeric “universal input” types NdioCounterPoint, NdioResistivePoint, NdioThermistorPoint, and NdioVoltagePoint. ([CounterInputPoint](#), [ResistiveInputPoint](#), [ThermistorInputPoint](#), [Voltage-InputPoint](#))
- **BooleanWritable**
For for boolean output type NdioBooleanWritable ([BooleanOutputWritable](#)).
- **NumericWritable**
For numeric output type NdioVoltageOutput ([VoltageOutputWritable](#)).

Each Ndio point type is typically unique by one or more properties in its *proxy extension*. Also, the proxy extension in each type contains the same (common) properties. Things unique to each Ndio point type are explained in following sections, as well as common properties in each point’s proxy extension.

Ndio Proxy Ext common properties

Figure 3-11 Common properties among Ndio proxy extensions



As shown in the property sheet view ([Figure 3-11](#)), each Ndio point has the following properties in its proxy extension (Proxy Ext):

- **Status**
(read only) Status of the proxy extension.
- **Fault Cause**
(read only) If point has fault status, provides text description why.
- **Enabled**
Either true (default) or false. While set to false, the point’s status becomes disabled and Ndio polling is suspended.
- **Device Facets**
(read only) Native facets used in proxy read or writes (Parent point’s facets are used in point status display, and are available for edit in Add and Edit dialogs in Point Manager).
- **Conversion**
Specifies the conversion used between the “read value” (in Device Facets) and the parent point’s output (in selected point facets). For details see [“Conversion types in Ndio proxy points”](#) on page 3-11.
- **Tuning**
Assigned tuning policy, in Ndio is typically “Default Policy.”
- **Read Value**
(read only) Last value read, using device facets.
- **Write Value**
(read only) Applies if writable point only. Last value written, using device facets.
- **Poll Frequency**
Assigned poll frequency group, either Slow, Normal (default), or Fast.
- **Point Type**
(read only) Ndio point type, such as “Resistive Input,” “Boolean Output,” and so on.

- **Address**
Point's I/O terminal address for its hardware type (UI, AO, DO), automatically found if added from Learn Mode discover. If address is set to out of range (relative to I/O board) or is a duplicate address (same address as same hardware type, same board), the point has a fault status.
- **Synced**
(read only) Either false or true. Reflects if the Ndio processor has synced with the current software configuration of the I/O point. Should always be true.
- **Raw**
(read only) Raw 12-bit unsigned integer value returned by the Ndio processor.

Conversion types in Ndio proxy points

In any Ndio proxy point a Conversion property is available, with the following available Ndio selections:

Note: Currently, Workbench provides no filtering of Conversion types based on the Ndio proxy point being configured. For example, you see the same Conversion drop-down selection list for a VoltageInputPoint as you do for a BooleanOutputWritable. However, only a few combinations are typically useful. In addition, there may be “non Ndio” conversion types included, such as “Sm2 Flow Conversion,” which can be ignored.

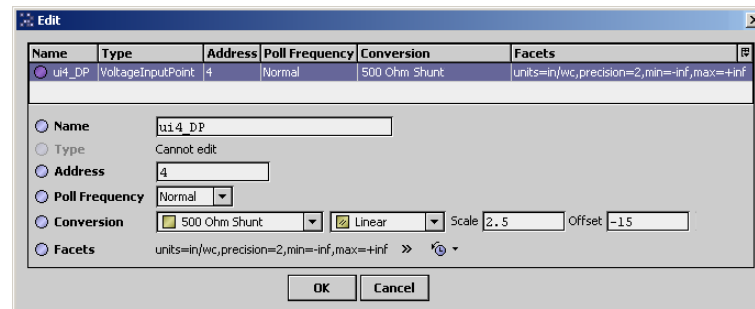
- **500 Ohm Shunt**
(Starting in AX-3.4 and in builds 3.3.28, 3.2.23, and 3.1.33) Applies only to a VoltageInputPoint used to read a 4-to-20mA sensor, where the UI input requires a 500 ohm resistor wired across (shunting) the input terminals. The input signal is 2 to 10V. If this conversion type is not available, select Linear conversion instead.

As compared to a Linear or Generic Tabular conversion, this selection provides better resolution near the upper (20mA/10V) range of the input, compensating for input clamping protection the circuitry automatically applies when input voltage rises above 3.9V.

Note: Currently, selection of 500 Ohm Shunt produces a “secondary” selection for conversion type, with all conversion types shown again. Typically, only the Linear type is valid, providing the 4-20mA sensor has a linear response.

In the Scale and Offset fields for this Linear conversion, you enter whatever calculated values are needed to display in the appropriate units of measurement for that sensor. Figure 3-12 shows an example configuration of such a point.

Figure 3-12 Example VoltageInputPoint used for 4-to-20mA input (Edit dialog from Point Manager)



For background on this example configuration, see “Scale and offset calculation example 2”.

In the unusual case of a non-linear 4-20mA sensor, providing you are using AX-3.4 or later ndio and have the kitIo module installed, the secondary conversion type “Generic Tabular” can be used. For more details, see

- **Default**
(The default selection). Conversion between “similar units” is automatically performed within the proxy point. For example, for an ThermistorInputPoint if you set the units in its facets to degrees F, the point output automatically converts to the appropriate value, from degrees C. If you set the parent point's Facets to dissimilar units (often the case with a VoltageInputPoint), the parent point has a fault status to indicate a configuration error. In this case, you must select Linear conversion, and also set “Units” in the point's “LinearCalibrationExt” to match the point's facets. See “Scale and offset calculation (linear)” on page 3-12 and “Linear Calibration Ext” on page 3-14.
- **Generic Tabular**
(AX-3.4 and later only, requires kitIo module to be installed). This Conversion type allows non-linear support for devices other than for thermistor temperature sensors with units in temperature. Generic Tabular uses a “lookup table” method similar to the “Thermistor Tabular” conversion, but without predefined output units. Selection provides a popup Tabular Conversion Dialog in which you can enter a custom “source-to-result” non-linear curve, including the ability to import

and export response curve files. For more details, see “Non-linear sensor support” on page 3-13 and “Curve File Import/Export notes” on page 3-20.

- **Linear**
Applies to [VoltageInputPoint](#), [ResistiveInputPoint](#), and [VoltageOutputWritable](#) points. For these points, you typically want the point’s output value in some units other than Device Facets (voltage or resistance). The Linear selection provides two fields to make the transition:
 - Scale: Determines linear response slope.
 - Offset: Offset used in output calculation.For related details, see “[Scale and offset calculation \(linear\)](#)” on page 3-12.
Note: For a [VoltageInputPoint](#) used for a 4-to-20mA sensor (shunt resistor on UI input), another primary conversion type should be used, if available: 500 Ohm Shunt, with Linear typically selected as the “secondary” conversion. See more about that other conversion type above.
- **Reverse Polarity**
Useful in a [BooleanInputPoint](#) point or [BooleanOutputWritable](#) to reverse the logic of the hardware binary input or output.
Note: Be careful in the use of the reverse polarity conversion, as it may lead to later confusion when troubleshooting logic or communications problems.
- **Thermistor Type 3**
Applies to a [ThermistorInputPoint](#) point, where this selection provides a “built-in” input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.
- **Tabular Thermistor**
Applies to a [ThermistorInputPoint](#) point, where this selection provides a control for a popup dialog for a custom resistance-to-temperature value response curve, including ability to import and export curve response files. For more details, see “[Tabular Thermistor notes](#)” on page 3-19 and “[Curve File Import/Export notes](#)” on page 3-20.

Scale and offset calculation (linear)

Unless you want a [VoltageInputPoint](#) or [VoltageOutputWritable](#) to operate with facets (units) of volts, you must set the point’s facets to the desired units, and set the ProxyExt Conversion property to *Linear*. You must then enter your calculated scale and offset values in the Linear fields. Also, you must set the “Units” in its [Linear Calibration Ext](#) to match an *input* point’s facets—otherwise a fault status occurs.

Note: This also applies to any linearly-responding [ResistiveInputPoint](#) using point facets other than ohms.

Use the following formulas to calculate the Linear scale and offset values:

where: x1 and x2 are the Ndio values (e.g voltage), and y1 and y2 are the corresponding desired units.

- **Scale**
 $Scale = (y2 - y1) / (x2 - x1)$
- **Offset**
 $Offset = y1 - (Scale * x1)$

See [example 1](#) and [example 2](#).

Scale and offset calculation example 1

You have a 6-to-9Vdc actuator to control with a [VoltageOutputWritable](#), in terms of 0-to-100% input. Configure this point’s facets to have units: “misc,” “percent” (%).

Set the ProxyExt’s Conversion property to Linear, and enter these calculated scale and offset values:

- **Scale**
 $Scale = (y2 - y1) / (x2 - x1)$
 $Scale = (100\% - 0\%) / (9V - 6V) = 100/3 = 33.3333333$
- **Offset**
 $Offset = y1 - (Scale * x1)$
 $Offset = 0\% - (33.3333333 * 6V) = 0 - 200 = -200$

Scale and offset calculation example 2

You have a linear 4-to-20mA differential pressure sensor to read with a [VoltageInputPoint](#) (using an external 500 ohm resistor wired across the input), reading effectively 2V to 10V. The range of this sensor is from -10 in.wc. to +10 in.wc. Configure this point’s facets to have units: “pressure,” “in/wc”.

Set the ProxyExt’s Conversion property to use “500 Ohm Shunt” (if available), then “Linear” as secondary, and in the Linear fields enter these calculated scale and offset values:

- **Scale**
 $Scale = (y_2 - y_1) / (x_2 - x_1)$
 $Scale = (+10in.wc - -10in.wc) / (10V - 2V) = 20/8 = 2.5$
- **Offset**
 $Offset = y_1 - (Scale * x_1)$
 $Offset = -10in.wc - (2.5 * 2V) = -10 - 5 = -15$

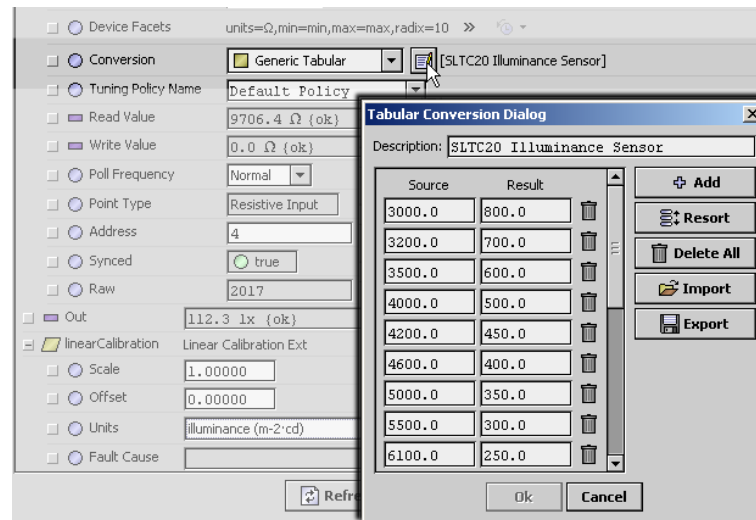
Figure 3-12 shows this point’s configuration in a Workbench dialog. Note that the Linear Calibration Ext for this point will require its “Units” property to be configured to match the facets for this point—otherwise, the point will have a fault status. See the “Linear Calibration Ext” section for related details.

Non-linear sensor support

In ndio AX-3.4 and later, providing that the **kitIo** module is installed, the conversion selection **Generic Tabular** provides a way for a **VoltageInputPoint** or **ResistiveInputPoint** to support a *non-linear* signal on a universal input (using point Facets *other* than temperature). Note that the **ThermistorInputPoint** also provides non-linear support for resistance-based temperature sensors, but point Facets are limited to temperature.

If you select Conversion type of “Generic Tabular” an edit control (note pad icon) appears in the property sheet beside it. Click the icon to see the **Tabular Conversion Dialog**, as shown in Figure 3-13.

Figure 3-13 Tabular Conversion Dialog from edit control



This dialog allows you to edit the current “source-to-results” non-linear curve used by the proxy point, import another curve (.xml file), or export (save) the current curve as an .xml file. In this way you can provide any custom non-linear curve needed. Figure 3-13 shows a point with a non-linear curve described in the “Non-linear sensor example” on page 3-14.

The following notes apply to using this feature:

- A curve requires at least 2 points (rows), each using “Source” to “Results” values.
 - Source values use the “Device Facets” of the Nrio proxy point, and must be in the range of the controller input. Therefore, if a **VoltageInputPoint**, source values must be between 0 to 10 (Volts), or if a **ResistanceInputPoint**, source values must be between 0 to 100000 (Ohms).
 - Result values are typically the *measured value*, regardless of units, at each source point. This assumes no further scaling using a **LinearCalibrationExt**—i.e., its **Scale** property is the default “1”.
- Points are used in ascending order, by the Source column.
- You can add as many points as is needed (there is no hard-coded limit). Click the **Add** button to add a new row for a point.
- If you skip a point, just add it at the end, and then click **Resort**. This automatically reorders all points in ascending order, by the Source column.
- Click the delete icon (trash can) beside any point to delete it from the curve. If you click the **Delete All** button, all points (plus any Description text) is removed (typically, you do this only to enter a *new* non-linear curve).
- Description text appears in the ProxyExt property sheet, beside the Conversion edit icon.
- When you click **Save**, the curve configuration is stored as part of the conversion object in the station (there is no association with any external file, in case you imported a non-linear curve).

Note: If you update a non-linear curve (say, add a point) after it was imported in one or more Nrio points, you need to export it (save) and re-import it in other points, in order for them to be updated.

Related to this, see “Curve File Import/Export notes” on page 3-20.

Non-linear sensor example

You have a photoresistor-based illuminance sensor that measures light output from 0 to 800 lux, supplying a resistance of 30K ohms to 3K ohms. Using a [ResistiveInputPoint](#), select the conversion type **Generic Tabular**, and click the edit control for the **Tabular Conversion Dialog** (see [Figure 3-13](#) on page 13).

In the dialog, enter the sensor’s non-linear response curve (shown below). Configure this point’s facets to have units: “illuminance,” “lux” (lx), and the same units in the point’s [Linear Calibration Ext](#).

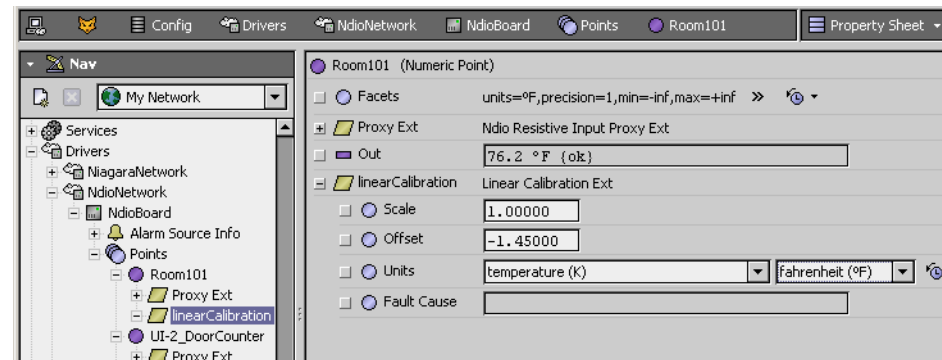
Table 3-1 Example illuminance sensor source (ohms) to results (lux)

Ohms	Lux
3000	800
3200	700
3500	600
4000	500
4200	450
4600	400
5000	350
5500	300
6100	250
6900	200
8200	150
10200	100
10900	90
11600	80
12400	70
13300	60
14400	50
15800	40
17700	30
20300	20
30000	0

Linear Calibration Ext

By default, three of the UI-type Ndio points ([ResistiveInputPoint](#), [ThermistorInputPoint](#), [VoltageInputPoint](#)) include a “linearCalibration” slot containing 4 properties, as shown in [Figure 3-14](#).

Figure 3-14 Linear Calibration Extension



These properties allow you to “calibrate” the calculated value before is applied to the Out slot, where: $[(\text{calculatedValue} \times \text{Scale}) + \text{Offset}] = \text{Out value}$.

Usage is optional, although Offset and Units are commonly configured.

Note: *In most cases where the parent Ndio proxy point's facets have been edited from defaults, note you must edit the Units value in the Linear Calibration Ext to match the units in the point facets, otherwise the parent proxy point will have a fault status!*

Typically, you see this fault status immediately after you add a new input point, for example a VoltageInputPoint or ResistanceInputPoint, and configure it with a Linear conversion type (including a scale and offset), and then specify the point's facets. It may not be immediately clear that the problem is in this Linear Calibration Ext, where you must match its Units value to the units in the point's facets.

- **Scale**
Default is 1.0. Typically, you leave this at default. One exception is if you copied the LinearCalibrationExt under a CounterInputPoint, in order to get a scaled total.
- **Offset**
Default is 0.0. Can be either a positive or negative value, as needed. Often, this is useful to compensate for signal errors introduced by sensor wiring resistance.
- **Units**
Set this to the same units as in the parent proxy point's facets (see [Note](#): above).
- **Fault Cause**
Available in AX-3.2 and later. Provides a reason whenever this extension causes the parent proxy point to be in fault, e.g.: Units between point and extension are not convertible.

Note: *You can also copy the linearCalibration extension without error to other Nrio points based on Numeric points. For example, a CounterInputPoint where a "scaled total" out value is desired. Consider a CounterInputPoint for a flow rate meter that provides a contact closure for every 0.15 gallons, configured as:*

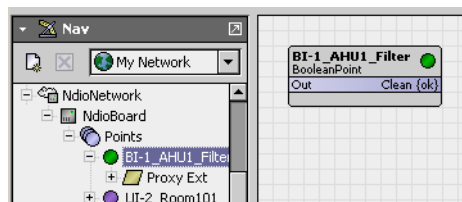
- Facets: gallons [units: volume (m³), gallon (gal)]
- NrioCounterInputProxyExt:
 - Conversion: Default
 - Output Select: Count
 - Rate Calc Type: FixedWindowRateType
 - Rate Calc:
 - Scale: 9.0
 - Interval: 15s
- LinearCalibrationExt:
 - Scale: 0.15
 - Offset: 0.0

*By copying a LinearCalibrationExt to the CounterInputPoint and specifying the item quantity/pulse as the Scale value (in this case, 0.15), the count output value will read in gallons, rather than number of pulses. Note that the "rate calc" setup provides the "gallons per minute" scaling that will reflect in the Rate slot of the point's ProxyExt, where Scale was calculated as 60 (sec/min) * 0.15 gal/pulse = 9.*

BooleanInputPoint

A BooleanInputPoint is a BooleanPoint with NdioBooleanInputProxyExt. It configures a UI to read contact closures as a status boolean (equipment status, binary input, BI).

Figure 3-15 Ndio Boolean Input Point



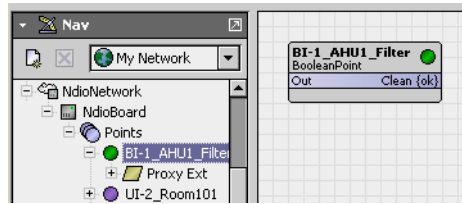
The BooleanInputPoint contains only [common Ndio Proxy Ext properties](#). Among these, note that the "Device Facets" property is blank by default.

Note: *The default operation is normal logic (closed contacts at UI is output value "true"). If needed, you can "flip" this logic in the ProxyExt by assigning a Conversion type of "Reverse Polarity." This causes the "reversed" boolean state at the output value. Typically, you monitor normally open (N.O.) equipment contacts using normal logic.*

BooleanOutputWritable

A BooleanOutputWritable is a BooleanWritable with NdioBooleanOutputProxyExt. It represents a digital output (DO). Depending on I/O platform, this may be a relay output or triac output. All standard BooleanWritable features apply, including right-click *actions*, priority input scheme, and minimum on/off properties (for details, see “About writable points” in the *User Guide*).

Figure 3-16 Ndio Boolean Output Writable



Note: The default operation is normal logic (closed contacts at DO if input value “true”). If needed, you can “flip” this logic in the ProxyExt by assigning a Conversion type of “Reverse Polarity.” This causes the “reversed” boolean state going from input to output.

In addition to the common Ndio Proxy Ext properties, the following additional properties are also available, contained under a **Hardware Override** container slot:

Note: The Hardware Override properties apply only to an I/O hardware module that was never released into production. Therefore, you can safely ignore the following properties under the Hardware Override slot:

- **Overridden**
(read only, transient) Either false or true, as to whether the hardware override switch is active. Uses Baja data type Boolean.
- **Fault Enable**
Either false or true (default). Enables a hardware override to mark the parent point in fault (while overridden is true).
- **Alarm Enable**
Either false (default) or true. Enables a hardware override to mark the parent point in alarm (while overridden is true).
Note: If you configure both Fault Enable and Alarm Enable true, a fault alarm is generated upon an active hardware override.
- **Alarm Class**
The alarm class to use for a hardware override-generated alarm or fault.

CounterInputPoint

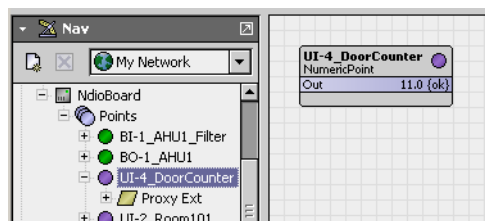
A CounterInputPoint is a NumericPoint with NdioCounterInputProxyExt. It configures a UI to count dry-contact pulses up to 20 Hz, as well as calculate a numeric rate. In the ProxyExt, you specify which value is to appear at the proxy point’s Out slot (either Count or Rate) as a status numeric.

The proxy extension contains configuration properties for rate calculation. It also contains a status property for total number of pulses counted (since it was last set or reset).

Note: Previous versions of this document included a note here describing how the I/O processor accumulates the count independently of the main JACE/NPM processor, stating that count could effectively continue during a station shutdown, or during much of the JACE boot process—picking up the count at the previous value upon reboot. However, this is inaccurate, as the I/O processor resets the count to 0 whenever communications to the station have been lost longer than 30 seconds (typically a reboot and subsequent station restart takes much longer than that).

Therefore, the count value is maintained by the running JACE station, and not the I/O processor. This means that pulses received during periods of station shutdown or station startup are not included in count.

Figure 3-17 Ndio Counter Input Point



The following sections apply to the CounterInputPoint:

- [About Ndio rate calculation](#)
- [Properties](#)
- [Actions](#)

About Ndio rate calculation

Configuration [properties](#) of the CounterInputPoint's Proxy Ext specify which of three Ndio rate calculators are used, either:

- [FixedWindowRateType](#)
- [SlidingWindowRateType](#)
- [TriggerTypeRate](#)

All three have same purpose: to calculate a pulse rate and update the point's output to reflect the newly calculated value. Each calculator is triggered to perform in a slightly different way.

FixedWindowRateType This rate calculator waits for the interval defined under the Rate Calc slot to elapse, then the recalculates the rate based on that interval.

Fixed window has 2 configurable rate calc values:

- **Scale**
A multiplier to use for adjusting the reported output value. See "[Properties](#)", Rate Calc, Scale.
- **Interval**
The amount of time between rate calculations.

SlidingWindowRateType This rate calculator is similar to the fixed window rate calculator. It also calculates based on the specified interval. However, it performs the calculation every interval/window seconds. This allows the rate to be updated more frequently while still maintaining that the calculation is based upon the specified interval. It then the recalculates the rate based on that interval.

Sliding window uses 3 configurable rate calc values:

- **Scale**
A multiplier to use for adjusting the reported output value. See "[Properties](#)", Rate Calc, Scale.
- **Interval**
The amount of time between rate calculations.
- **Windows**
The number of times during the interval that a recalculation is performed.

TriggerTypeRate This rate calculator is the simplest. It simply adds a recalculateRate *action* to the parent point. Typically, you link the action to the output of a TriggerSchedule configured at a specific interval (e.g. at 0, 15, 30, and 45 minutes every hour). The calculator performs its recalculation based on the interval between now and last trigger.

Trigger type use only one configurable rate calc value:

- **Scale**
A multiplier to use for adjusting the reported output value. See "[Properties](#)", Rate Calc, Scale.

Properties

Note: *It is recommended to leave the Conversion at "Default" in the NdioCounterInputProxyExt, so as not to interfere with rate calculation. If a scaled Count output is needed, add a LinearCalibrationExt to the point, and enter the item quantity/pulse as the Scale value there. See "[Linear Calibration Ext](#)" on page 3-14, including the ending [Note](#): about this application.*

In addition to [common Ndio Proxy Ext properties](#), the following additional properties are available in the NdioCounterInputProxyExt:

- **Output Select**
Specifies whether count total (Count) or count rate (Rate) is at the Out slot, as a status numeric. Default is Count.
- **Total**
(read only) Total number of pulses counted since the Proxy Ext was last set or reset. Data type is Baja Long.
- **Rate**
(read only) Calculated rate, based upon Rate Calc configuration. Data type is Baja Double.
- **Rate Calc Type**
Selectable to one of three types provided in the *ndio* module:
 - **FixedWindowRateType (default)**

- **SlidingWindowRateType**
- **TriggerRateType**

Note: Future rate calculators may be developed by third-parties. When this occurs, they will be available (if that module is installed), as pull-down menu options other than ndio.

- **Rate Calc**
Contains from one to three properties used in rate calculation, according to the selected rate calc type (see “About Ndio rate calculation” on page 3-17):
 - **Scale**
Default is value is 1. Appropriate value depends on the item quantity/pulse and desired rate units. Two examples are provided:
 - **Power Meter**
Each meter pulse indicates 0.15 kWh, and desired rate units is kW.
Scale = 3600 (sec/hr) * 0.15 kWh/pulse = 54
 - **Liquid Flow Meter**
Each meter pulse indicates 0.375 liters, and desired rate units is liters/minute (L/min).
Scale = 60 (sec/min) * 0.375 liters/pulse = 22.5
 - **Interval**
(not available if TriggerRateType) Default is value is 1 minute.
 - **Windows**
(available *only* if SlidingWindowRateType) Default is value is 6.
- **Rate Calc Time**
(read only) Reflects timestamp of last rate calculation.

Actions

The NdioCounterInputProxyExt has the following available actions:

- **Reset**
Sets the point’s Total (and Out value, if outputSelect is Count) to zero (0).
- **Set**
Sets the point’s Total (and Out value, if outputSelect is Count) to a specified count. Note this is an unscaled (actual number of pulses) value, so if a scaled Count is configured, it will show differently.

ResistiveInputPoint

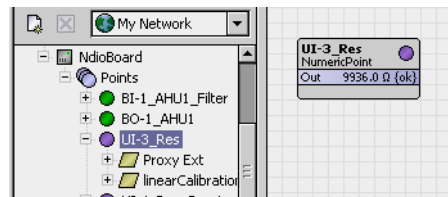
A ResistiveInputPoint is a NumericPoint with NdioResistiveInputProxyExt. It configures a UI to read a resistance from 0 to 100,000 ohms. Configuration provides for default (ohms).

- If a linear-responding device and other units are needed, select Conversion type *Linear*—see “Scale and offset calculation (linear)” on page 3-12.
- If a non-linear device and other units are needed, if the JACE is running AX-3.4 (with `kitIo` module installed), select Conversion type *Generic Tabular*—see “Non-linear sensor support” on page 3-13.

Note: If using a thermistor temperature sensor, see [ThermistorInputPoint](#).

The ResistiveInputPoint is pre-configured with a linearCalibration extension, to allow for offset correction. For details, see “Linear Calibration Ext” on page 3-14.

Figure 3-18 Ndio Resistive Input Point



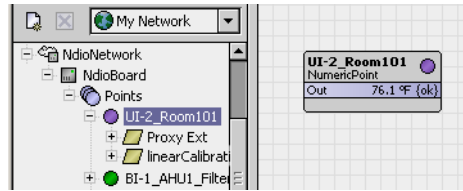
The ResistiveInputPoint’s Proxy Ext contains only [common Ndio Proxy Ext properties](#). Among these, note that the read-only “Device Facets” property is preset to ohms.

ThermistorInputPoint

A ThermistorInputPoint is a NumericPoint with NdioResistiveInputProxyExt configured for a UI to read a thermistor temperature sensor. It is pre-configured with a linearCalibration extension, for offset correction. For details, see “Linear Calibration Ext” on page 3-14.

Note: A *ThermistorInputPoint* is the same as a *ResistiveInputPoint*, only with a different “Conversion” setting in its *ProxyExt*. However, the *Ndio Point Manager* and *ndio palette* both list this point separately, so it is covered here separately.

Figure 3-19 Ndio Thermistor Input Point



The point’s *Proxy Ext* contains [common Ndio Proxy Ext properties](#). Among those properties, note that the read-only “Device Facets” is preset to ohms.

Depending on the type of thermistor temperature sensor you are using, you can select a Conversion type of either Type 3 Thermistor, or any other using a [Tabular Thermistor](#) conversion.

Type-3 Thermistor

[Table 3-2](#) shows the hard-coded response curve used if *ProxyExt* property Conversion is set to “Thermistor Type 3”:

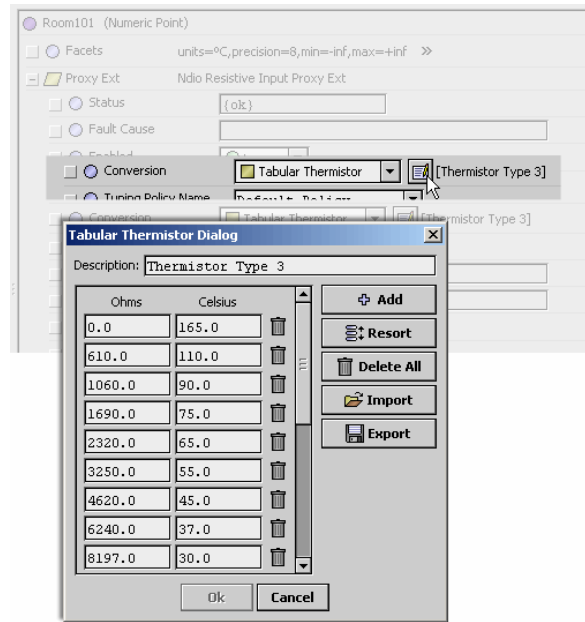
Table 3-2 Type-3 Thermistor Ndio Curve

Ohms	Deg. C
0	165
610	110
1060	90
1690	75
2320	65
3250	55
4620	45
6240	37
8197	30
10000	25
12268	20
15136	15
18787	10
23462	5
29462	0
37462	-5
47549	-10
61030	-15
78930	-20
100000	-25

Tabular Thermistor notes

In the *ProxyExt* of the thermistor point, if you select Conversion type of “Tabular Thermistor,” an edit control (note pad icon) appears in the property sheet beside it. Click the icon to see the Tabular Thermistor dialog, as shown in [Figure 3-20](#).

Figure 3-20 Tabular Thermistor Dialog from edit control



This dialog allows you to edit the current ohms-to-degrees Celsius curve used by the proxy point, import another thermistor curve (.xml file), or export (save) the current thermistor curve as an .xml file. In this way you can provide any custom thermistor curve needed.

The following notes apply to using this feature:

- A curve requires at least 2 points (rows), each using ohms to degrees Celsius values.
- Points are used in ascending order, by the ohms column.
- You can add as many points as is needed (there is no hard-coded limit). Click the **Add** button to add a new row for a point.
- If you skip a point, just add it at the end, and then click **Resort**. This automatically reorders all points in ascending order, by the ohms column.
- Click the delete icon (trash can) beside any point to delete it from the curve.
- If you click the **Delete All** button, all points (plus any Description text) is removed (typically, you do this only to enter a new thermistor curve).
- Description text appears in the ProxyExt property sheet, beside the Conversion edit icon.
- When you click **Save**, the curve configuration is stored as part of the conversion object in the station (there is no association with any external file, in case you imported a thermistor curve).

Note: If you update a thermistor curve (say, add a point) after importing it into one or more Ndio points, you need to import it in those points again, in order for them to be updated.

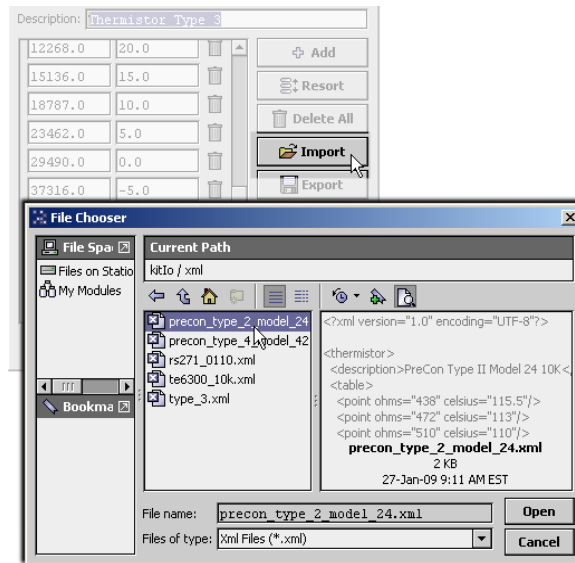
Related to this, see [Curve File Import/Export notes](#).

Curve File Import/Export notes

As needed, import (load) or export (save) a thermistor or non-linear *curve file* in xml format using the **Import** and **Export** buttons in either the **Tabular Thermistor Dialog** (Figure 3-20) for a “Tabular Thermistor” conversion, or **Tabular Conversion Dialog** (Figure 3-13 on page 13) for a “Generic Tabular” conversion.

When you do this, the standard Niagara **File Chooser** dialog appears, as shown in [Figure 3-21](#).

Figure 3-21 File Chooser to locate thermistor or non-linear curve .xml file



To *import* a thermistor curve file, you can navigate under your “My Modules” folder to the xml folder in the **kitIo** module (Jar) file (if available, see [Figure 3-21](#)). If the kitIo module is not available, navigate to the xml folder in the **ndio** module (Jar) file.

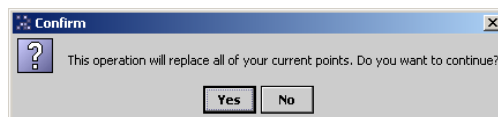
Or, you can navigate to any other location you have a thermistor curve file. Import any other saved, custom non-linear conversion .xml files the same way—see “[Non-linear sensor support](#)” on page 3-13.

As of the date of this document, the **kitIo** module contains an xml folder with five (5) different thermistor curves in xml format, as follows:

- precon_type_2_model_24.xml — For Precon type 2 model 24 sensor
- precon_type_4_model_42.xml — For Precon type 4 model 42 sensor
- rs271_0110.xml — For Radio Shack sensor model 271-0110
- te6300_10k.xml — For TE-6300 10K type sensor
- type_3.xml — For standard Type-3 Thermistor sensor

Note: Upon any curve file import, all existing thermistor or non-linear curve configuration for that Ndio proxy point is overwritten (replaced by configuration in the imported file). A confirmation dialog advises you of this before you import, as shown in [Figure 3-22](#)

Figure 3-22 Curve file import confirmation



To *export* a curve .xml file, use the **File Chooser** to navigate to any location you wish to save the current thermistor curve configuration, and supply a File Name. You can use this saved .xml file in the configuration of any other like Ndio proxy point, whether in this station or another station.

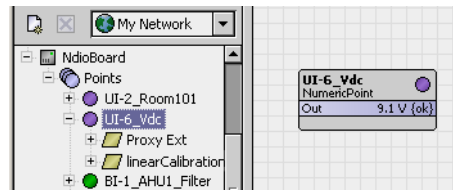
VoltageInputPoint

A VoltageInputPoint is a NumericPoint with NdioVoltageInputProxyExt. It configures a UI to read a Vdc signal from 0-to-10V. Currently, configuration provides for reading in volts or a linear response in other units, by selecting “Linear” as the conversion type. See “[Scale and offset calculation \(linear\)](#)” on page 3-12.

Note: Besides reading a 0-10Vdc sensor, this point is also used for a 4-20mA sensor, where an external 500 ohm resistor is wired across the UI terminals. In this case only, select the “500 Ohm Shunt” as the point’s conversion type. See “[Conversion types in Ndio proxy points](#)” on page 3-11 for more details.

The VoltageInputPoint is pre-configured with a linearCalibration extension, to allow for offset correction. For details, see “[Linear Calibration Ext](#)” on page 3-14.

Figure 3-23 Ndio Voltage Input Point

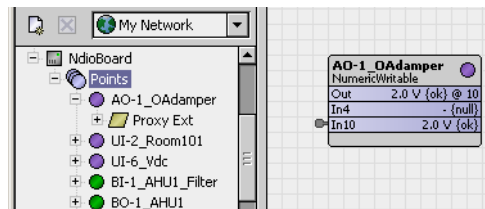


The VoltageInputPoint’s Proxy Ext contains only [common Ndio Proxy Ext properties](#). Among these, note that the read-only “Device Facets” property is preset to volts (V).

VoltageOutputWritable

A VoltageOutputWritable is a NumericWritable with NdioVoltageOutputProxyExt. It represents a 0-to-10Vdc analog output (AO). All standard NumericWritable features apply, including right-click *actions* and priority input scheme (see “[About writable points](#)” in the *User Guide*).

Figure 3-24 Ndio Voltage Output Writable



The point’s Proxy Ext contains [common Ndio Proxy Ext properties](#). Among those properties, note that the read-only “Device Facets” is set to units Volts (V). For operation in units other than volts, see “[Scale and offset calculation \(linear\)](#)” on page 3-12.

In addition to the [common Ndio Proxy Ext properties](#), the following additional properties are also available, contained under a **Hardware Override** container slot.

Note: *The following four Hardware Override properties were intended for a I/O hardware module that was never released with NiagaraAX support. Therefore, they are not functional and can be safely ignored.*

- **Overridden**
(read only, transient) Either false or true, as to whether the hardware override switch is active. Uses Baja data type Boolean.
- **Fault Enable**
Either false or true (default). Enables a hardware override to mark the parent point in fault (while overridden is true).
- **Alarm Enable**
Either false (default) or true. Enables a hardware override to mark the parent point in alarm (while overridden is true).

Note: *If you configure both Fault Enable and Alarm Enable true, a fault alarm is generated upon an active hardware override.*

- **Alarm Class**
The alarm class to use for a hardware override-generated alarm or fault.

CHAPTER 4

Ndio Plugin Guides

Plugins provide *views* of components, and can be accessed many ways—for example, double-click a component in the tree for its *default* view. In addition, you can right-click a component, and select from its **Views** menu. For summary documentation on any view, select **Help > On View (F1)** from the Workbench menu, or press F1 while the view is open.

Ndio Plugin Guides Summary

Summary information is provided on views specific to components in the `ndio` module, as follows:

- [NdioBoardManager](#)
- [NdioPointManager](#)

ndio-NdioBoardManager

Use the NdioBoardManager to add, edit, and access NdioBoards that represent physical modules of I/O on the host JACE controller platform. The NdioBoardManager is a view on the [NdioNetwork](#). To view, right-click [NdioNetwork](#) and select **Views > Ndio Board Manager**.

Tables in the NdioBoardManager view have the following columns:

- **Name**
Component name of the I/O board
- **Type**
Niagara type of the board (currently there is only NdioBoard)
- **Exts**
Shortcut to the device extensions (only one extension, Points).
- **IoPort**
Communication port to which the I/O board is connected.
- **Multiple Processors**
Indicates whether this I/O board has more than one processor.
- **Firmware Version**
The I/O processor firmware revision for each of the I/O board's processors.
- **Config Code**
Manufacturing numerical code that identifies the I/O board's feature set.
- **UiCount**
Number of universal inputs (UIs) available on the I/O board.
- **BoCount**
Number of boolean outputs (BOs) available on the I/O board.
- **AoCount**
Number of analog outputs (AOs) available on the I/O board.
- **Status**
Current status of the I/O board.

For more details, see “[Ndio Board Manager](#)” on page 3-4.

ndio-NdioPointManager

Use the NdioPointManager to add, edit, and access Ndio proxy points under the [NdioPointDeviceExt](#) extension of a selected [NdioBoard](#), or in an [NdioPointFolder](#). The NdioPointManager is the default view on both of those components. To view, right-click the [NdioPointDeviceExt](#) extension or [NdioPointFolder](#) and select **Views > Ndio Point Manager**.

Tables in the NdioPointManager view have the following columns:

- **Name**
Component name of the Ndio proxy point.
- **Type**
Ndio proxy extension type of the configured point.
- **Io Type**
Generic Ndio type of a discovered point.
- **Address**
Hardware I/O address of the point.
- **Out**
Current value of the point.

For more details, see [“Ndio Point Manager”](#) on page 3-7.

CHAPTER 5

Ndio Component Guides


These component guides provides summary help on Ndio components.

Ndio Component Reference Summary


Summary information is provided on components in the `ndio` module, listed alphabetically as follows:

- [FixedWindowRateType](#)
- [LinearCalibrationExt](#)
- [NdioBoard](#)
- [NdioBoardFolder](#)
- [NdioBooleanInputProxyExt](#)
- [NdioBooleanOutputProxyExt](#)
- [NdioCounterInputProxyExt](#)
- [NdioHardwareOverride](#)
- [NdioNetwork](#)
- [NdioPointDeviceExt](#)
- [NdioPointFolder](#)
- [NdioPollScheduler](#)
- [NdioResistiveInputProxyExt](#)
- [NdioVoltageInputProxyExt](#)
- [NdioVoltageOutputProxyExt](#)
- [SlidingWindowRateType](#)
- [ThermistorType3Type](#)
- [TriggerRateType](#)


ndio-FixedWindowRateType

 Contains rate calculation parameters for the parent [NdioCounterInputProxyExt](#). For details, see “[FixedWindowRateType](#)” on page 3-17 and “[About Ndio rate calculation](#)” on page 3-17.


ndio-LinearCalibrationExt

 [LinearCalibrationExt](#) is an point extension that allows for calibration adjustment of an input to a known measured value. For more details, see “[Linear Calibration Ext](#)” on page 3-14.

ndio-NdioBoard

 [NdioBoard](#) represents an Ndio board that can contain different I/O points. You use the [NdioPoint-Manager](#) view of its *child* [NdioPointDeviceExt](#) extension to discover, add, and edit Ndio proxy points. For more details, see “[About the NdioBoard](#)” on page 3-5.


ndio-NdioBoardFolder

 [NdioBoardFolder](#) is the Ndio implementation of a folder under an [NdioNetwork](#). Usage is optional. Each [NdioBoardFolder](#) has its own [NdioBoardManager](#) view.


You can use the New Folder button in the [NdioBoardManager](#) view to add an [NdioBoardFolder](#). It is also available in the `ndio` palette.

Note: *Typically, there is little need for NdioBoardFolders, even if the Niagara host platform has multiple physical (Ndio) I/O modules.*


ndio-NdioBooleanInputProxyExt

 [NdioBooleanInputProxyExt](#) is the proxy extension for an [NdioBooleanPoint](#). It represents an Ndio *boolean universal input*. For details, see “[BooleanInputPoint](#)” on page 3-15.

ndio-NdioBooleanOutputProxyExt

 NdioBooleanOutputProxyExt is the proxy extension for an NdioBooleanWritable. It represents an Ndio *boolean output*. For details, see “[BooleanOutputWritable](#)” on page 3-16.


ndio-NdioCounterInputProxyExt

 NdioCounterInputProxyExt is the proxy extension for an NdioCounterPoint. It represents an Ndio universal input configured for pulse-count and rate determination from dry contacts. Rate configuration is selectable as either:


- Fixed window type
- Sliding window type
- Trigger type

Additional rate parameters must be entered in the selected child component, such as [FixedWindowRateType](#). For more details, see “[CounterInputPoint](#)” on page 3-16.


ndio-NdioHardwareOverride

 NdioHardwareOverride is a container slot in the ProxyExt for [BooleanOutputWritable](#) and [VoltageOutputWritable](#) points. Child properties determine how hardware (switch) overrides are handled—and apply only to I/O modules that are equipped with hardware override switches.

ndio-NdioNetwork


 NdioNetwork represents a network of manageable Ndio boards. The [NdioBoardManager](#) is a view on the [NdioNetwork](#). The NdioNetwork is available in the ndio palette. For details, see “[About the Ndio Network](#)” on page 3-2.

ndio-NdioPointDeviceExt


 NdioPointDeviceExt (default name `Points`) is the container for Ndio proxy points under an [NdioBoard](#). Ndio proxy points represent Ndio data values. The default and primary view for the NdioPointDeviceExt is the [NdioPointManager](#). Apart from this view, Points serves only as the top container for all Ndio proxy points under that NdioBoard (it has no other properties).

The NdioPointDeviceExt for an NdioBoard is automatically created when you add an NdioBoard. For details, see “[Ndio Point Manager](#)” on page 3-7.


ndio-NdioPointFolder

 NdioPointFolder is an optional container for Ndio proxy points. The NdioPointFolder is available in the ndio module.


ndio-NdioPollScheduler

 NdioPollScheduler. The NdioPollScheduler is available in the ndio module. For details, see “[Ndio Network poll scheduler notes](#)” on page 3-4.


ndio-NdioResistiveInputProxyExt

 NdioResistiveInputProxyExt is the proxy extension for an NdioResistivePoint. It represents an Ndio *universal input* that reads a resistance (ohms) signal or a Thermistor temperature sensor. For more details, see “[ResistiveInputPoint](#)” on page 3-18 and “[ThermistorInputPoint](#)” on page 3-18.


ndio-NdioVoltageInputProxyExt

 NdioVoltageInputProxyExt is the proxy extension for an NdioVoltagePoint. It represents an Ndio *universal input* that reads a 0-to-10Vdc input signal, or a 4-to-20mA signal (with a 500 ohm resistor wired across the input terminals). For details, see “[VoltageInputPoint](#)” on page 3-21.


ndio-NdioVoltageOutputProxyExt

 NdioVoltageOutputProxyExt is the proxy extension for an NdioVoltageWritable. It represents an Ndio *analog output* (AO) that produces 0-to-10Vdc. For details, see “[VoltageOutputWritable](#)” on page 3-22.


ndio-SlidingWindowRateType

 Contains rate calculation parameters for the parent [NdioCounterInputProxyExt](#). For details, see “[SlidingWindowRateType](#)” on page 3-17 and “[About Ndio rate calculation](#)” on page 3-17.

ndio-ThermistorType3Type

 ThermistorType3Type is a thermistor type option for a [ThermistorInputPoint](#). For more details, see “[ThermistorInputPoint](#)” on page 3-18.

ndio-TriggerRateType

 Contains rate calculation parameters for the parent [NdioCounterInputProxyExt](#). For details, see “[TriggerTypeRate](#)” on page 3-17 and “[About Ndio rate calculation](#)” on page 3-17.