Technical Document

Niagara Drivers Guide

January 12, 2023

niagara4

Niagara Drivers Guide

Tridium, Inc.
3951 Westerre Parkway, Suite 350
Richmond, Virginia 23233
U.S.A.

Confidentiality

Trademark notice

Copyright and patent notice

Contents

About this guide

This topic contains important information about the purpose, content, context, and intended audience for this document.

Product Documentation

This document is part of the Niagara technical documentation library. Released versions of Niagara software include a complete collection of technical information that is provided in both online help and PDF format. The information in this document is written primarily for Systems Integrators. To make the most of the information in this book, readers should have some training or previous experience with Niagara software, as well as experience working with JACE network controllers.

Document Content

This document describes the basic driver architecture used in any NiagaraStation to retrieve and model real-time data values. The driver architecture includes parent network components containing one or more child device components, each with their respective device extension child components.

Document change log

Updates (changes and additions) to this document are listed below.

January 12, 2023

- Added new topic "HTML–5 Niagara Network Device Manager" to the "Plugins" chapter.
- Added "Https Cert" ("Alias", "Password", "Use global certificate password") and "Server Certificate Health" ("Requested Cert", "Returned Cert", "Cert Status") properties to "fox—FoxService" component.

October 1, 2022

Added new "Route To Station" property (Niagara 4.13) to "niagaraDriver-NiagaraFileImport" component and "New/Edit file import" window.

September 1, 2022

Added new topics to "Plugins" chapter

- Naxis Video Device Manager view
- HTML–5 Naxis Video Device Manager
- HTML-5 HTTP Client Device Manager
- HTML-5 Maxpro Device Manager
- HTML-5 Nsnmp Device Manager

June 27, 2022

Removed the "niagaraDriver-NiagaraFoxService" component from the "niagaraDriver" module section as it has been replaced by the "fox-FoxService" component in Niagara 4.0 and later.

Niagara 4.13: Added property "Route To Station" to component "niagaraDriver-NiagaraFileImport".

February 23, 2022

Added "HTML5 Niagara Point Manager View" to the "Plugins" chapter.

January 6, 2021

Reorganized, edited and created task topics. Updated for Niagara 4.10.

November 13, 2019

Added details on Conversion (Linear with unit) ProxyExt property.

June 4, 2019

Edited the TunnelService component details to reflect the latest changes for security. Updated FoxService component properties.

January 10, 2019

Minor correction in the procedure for "Viewing Virtual Px graphics on demand."

April 08, 2018

- In the section on , "Station Manager Notes", added a subtopic describing **Station Manager** enhancements for the System Indexing feature (in Niagara 4.6 and later).
- In the topic, "About persisting fetched tags", added a note clarifying that the `n:history` tag applies to local control points as well as to proxy points on the Supervisor.

Updated: March 09, 2018

- Added two procedures (draft version) in the Virtual Policies section, under "About virtualized Px views".

Updated: August 16, 2017

- Modified the topic, "About persisting fetched tags," adding content to clarify where to set the Persist Fetched Tag and related properties.

Updated: July 12, 2017

- Restored several legacy Users Extension and User Synchronization topics with the content (and some graphics) updated for Niagara 4

Updated: January 20, 2017

- Restored two topics on NiagaraNetwork's Virtual Policies, and the Virtual Policies Cache properties. Added the topic, About "virtualized" Px views, under the section on Virtual Policies, which provides information on properties new in Niagara 4.3

Initial Release: December 08, 2016

Related documentation

Every standalone driver guide relies on this guide for the explanation of common driver properties, concepts, and tasks.

- *Getting Started with Niagara (User Guide)*
- *Niagara Platform Guide*
- *Niagara Station Security Guide*
- *Niagara Alarms Guide*
- *Niagara Scheduling Guide*
- *Niagara Provisioning Guide*
- *Niagara System Database and System Indexing Guide*
- *Niagara Histories Guide*
- *Niagara LDAP Guide*

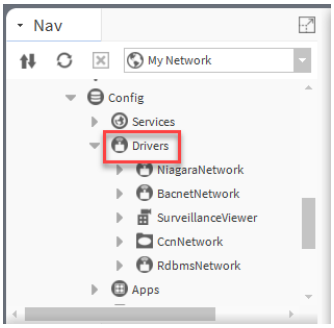
Chapter 1 Driver networks

Topics covered in this chapter

- ♦ Prerequisites
- ♦ About network architecture
- ♦ Table controls and options
- ♦ Adding a driver network
- ♦ About tuning policies
- ♦ Discovering objects
- ♦ Saving time when setting up identical objects
- ♦ Programming offline
- ♦ Group editing objects
- ♦ Tagging objects
- ♦ About communication components
- ♦ Driver troubleshooting

In any station, all driver networks are located under the **Drivers** container component in the station's database.

Figure 1 Drivers container in Nav tree



By default, stations include a **NiagaraNetwork** that can model the data from other subordinate stations. The **Drivers** container supports device networks, including Modbus, BACnet, Lonworks, and many others.

In addition to device networks, the navigation (Nav) hierarchy represented by **Config→Drivers→Driver→Device** supports features that are unrelated to devices. For example, the **FileNetwork** component found in the **driver** module treats history files as if they were devices for the purpose of importing and exporting them.

This chapter contains general information about the components and procedures that are common to all drivers.

Prerequisites

Before you configure the **NiagaraNetwork**, all hardware including devices should be installed and the driver that manages your remote controllers, devices and points should have been installed and configured.

The **NiagaraNetwork** has these requirements:

- All connections among Supervisor and remote controller stations are secure connections (https, foxs). Configuring any station for standard connections that are not secure is not recommended.
- To use a **FileNetwork**, the Supervisor station'sTridium license requires special licensing for the feature `fileDriver`. This feature may also contain a `history.limit=n` attribute, which defines the number of files that can be imported as histories (number of history file import descriptors).

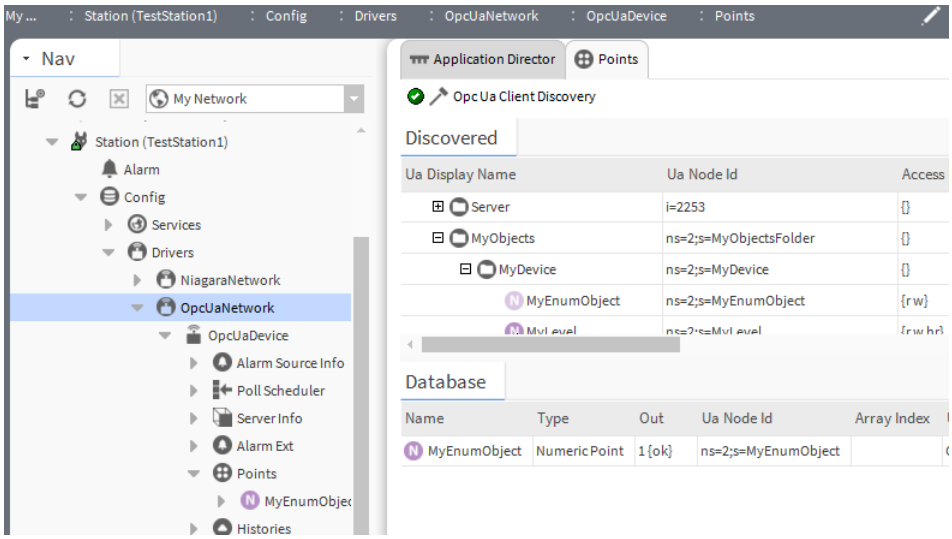
About network architecture

Niagara uses a consistent driver framework to represent each driver in a station database. This includes the upper-tier parent network component and one or more child device components, each with device ext (extension) child components.

Hierarchically, the component parentage is: network, device, device extensions, and points, as shown here. Using a network architecture enforces a logical structure and helps coordinate mechanisms used by the driver for communications. Exceptions to this rule are noted (as applicable) in documentation.

For drivers that use field bus communications, such as Lonworks, BACnet, and Modbus (among many others), the top-level network component corresponds directly to a physical network of devices. Often (except for BACnet), the network component matches one-to-one with a specific communication (comm) port on the host platform, such as a serial or RS-485 port, Lonworks FTT-10 port, or Ethernet port.

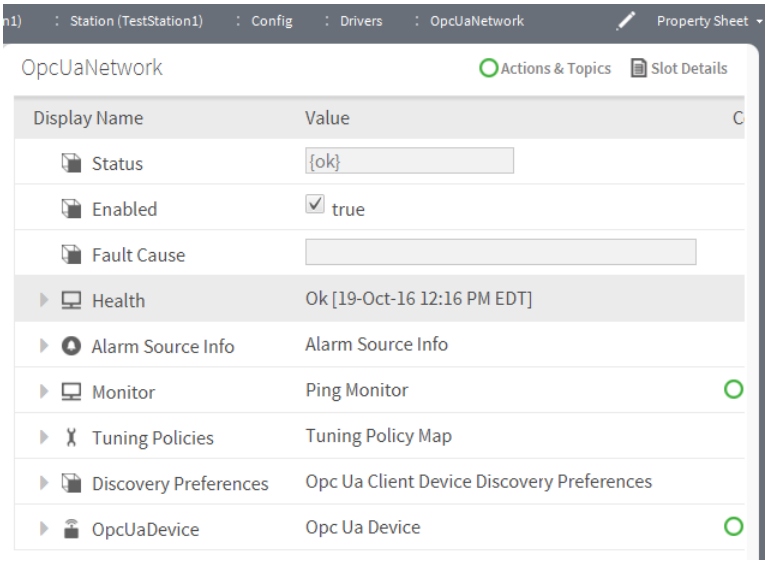
Figure 2 Example of a driver Device Manager view (OpcUa)



The driver network is the parent container for all device-level components. Devices list in tabular format in the default view for the network, the **DeviceManager**. You use the **Device Manager** to create and manage device components, and (if the driver provides this), use discover features.

Access these slots in the driver network's Property Sheet.

Figure 3 Example OpcUaNetwork property sheet



Depending on the specific driver, there may be additional slots and components in the driver network. For example, if the driver supports a field bus network, there may be **`PollScheduler`** and/or **`CommConfig`** slots. Some of these may require proper configuration before driver communications occur.

NOTE: A **BacnetNetwork** component is unique because it supports multiple logical BACnet networks, which sometimes use different comm ports (for example, a controller with BACnet MS/TP and one or more RS-485 ports).

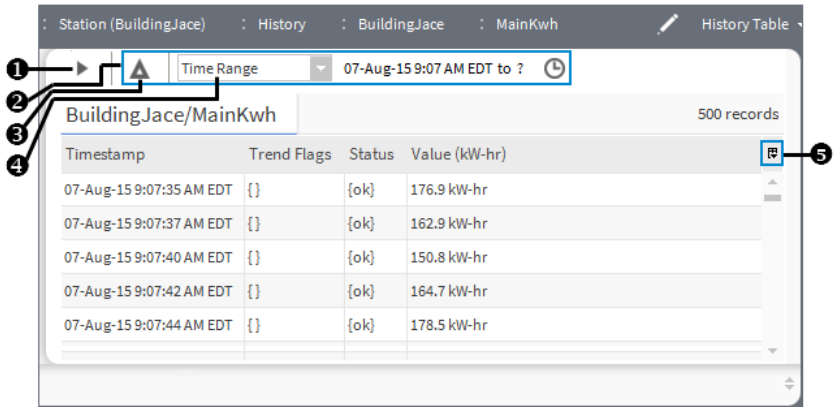
Other non-field-bus drivers also use a network architecture, for example the Nrio driver (Niagara Remote Input/Output) has a network (**NrioNetwork**) that interfaces with physical I/O points on the appropriate host controller or hardware I/O module (either directly attached or remotely connected). Database drivers use a network architecture, for example the **rdbSqlServer** driver includes an **RdbmsNetwork**. Database drivers apply only to Supervisor hosts.

Depending on the driver, other specific properties (or even component containers) may be found in the network component's property sheet.

Table controls and options

Many Workbench views present information in a table. All tables share similar features and controls.

Figure 4 Table controls and options



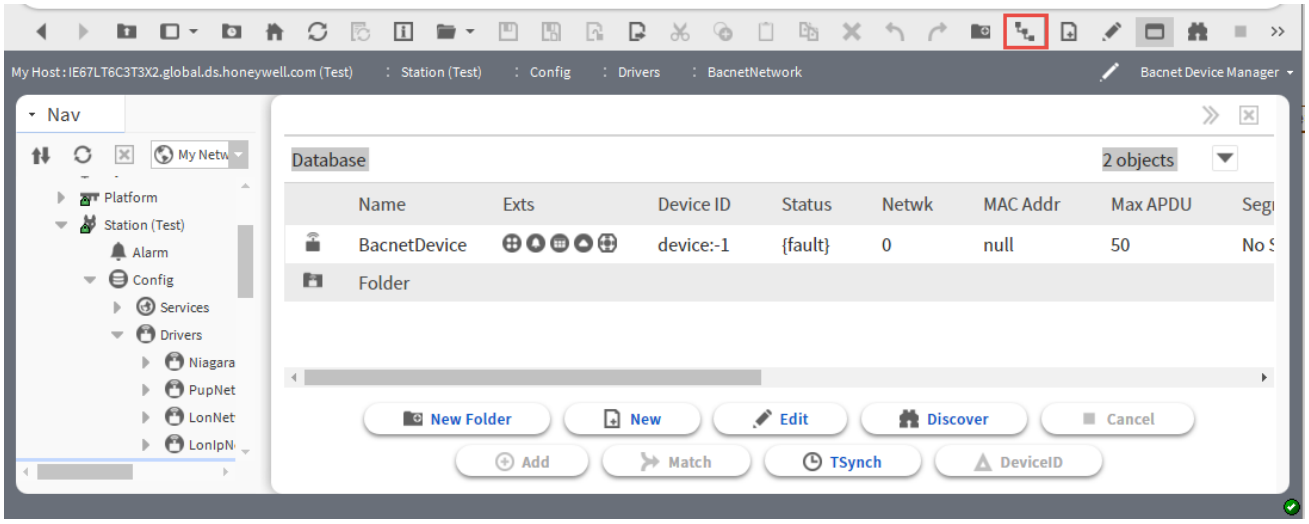
❶	**Live Updates** — The "play" icon, available for the **History Table** view, starts Live Updates (On Demand) updating. The icon changes to a "pause" icon while Live Updates is active.
❷	Data parameters — These controls include **Delta** (for history logging) and **Time Range** settings.
❸	**Delta** — Useful for history logging, displays value changes (delta) in your table.
❹	**Time range** — The drop down option list has a variety of predefined time range options, including an option that allows you to restrict your data presentation to a particular date and time range that you specify.
	Title bar — Displays the name of the data collection on the left side of the title bar and in some tables (collection table, history table, alarm extension manager, and others) displays the total number of records in the table on the right side of the title bar.
	Column headings — Each column of data has a title that indicates the data type.
	Column boundaries — Each column has a movable column boundary that can be used to re-size the column using the mouse control. Stretch or shrink column width by dragging the column boundary, as desired. Use the Reset column widths menu item to reset all column widths to their default size.
❺	**Table Options** — Located in the upper right corner of the table, the drop down list provides one or more of the following controls and options: • **Reset column widths** — sets all columns in the table to their default widths. This is useful if you manually changed widths of columns, and now some contents are hidden (even after scrolling). • **Export** — opens the **Export** window where you can choose to export the table to PDF, text, HTML, or CSV (comma separated variable). • Context-sensitive menu items — additional context-sensitive menu items appear depending on the component that you are viewing.

Device Manager tables

The **Device Manager** of a device folder is unaware of devices in other device folders (or in the root of the network component). However, from the root **Device Manager** view (of the network), you can flatten the folder structure to view all devices by clicking the **All Descendants** tool on the toolbar.

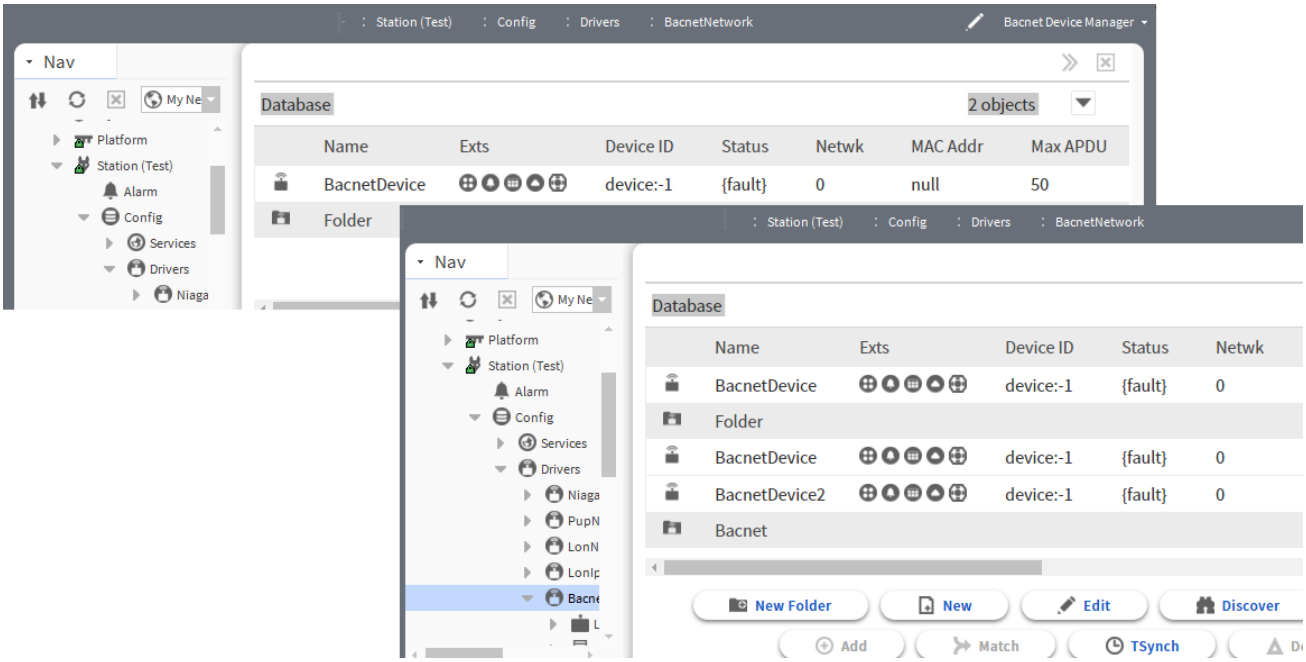
NOTE: In a large network, many subscribed device folders and devices, using the all descendants feature may decrease system performance.

Figure 5 All Descendants command



The result is a view that includes the devices managed by multiple device folders.

Figure 6 Device Manager showing multiple folders



If you are using device folders, and you click on a table column to restart devices, the view loses any device-to-device folder organization. However, you can always see the contents of device folders clearly in the Nav tree, and again when returning to the **Device Manager** view from another view.

Adding a driver network

Driver network setup in a remote controller station is basically the same for any driver. Examples include **LonNetwork**, **BacnetNetwork**, **OpcUaNetwork** and many more.

Prerequisites: You are working in Workbench and are connected to the Supervisor or remote controller station that will host the driver network.

If you are engineering a Supervisor (PC) station, the **NiagaraNetwork** may be the only network you need. Optionally, you may add a database network (providing the PC host is licensed for it). And, if a PC host is licensed as a direct integration Supervisor (for example, a BACnet Supervisor), you may add a driver network (for example, **BacnetNetwork**) directly under the Supervisor station's **Drivers** container.

Regardless of host platform, in most cases the network component is the only item you need to manually copy from a driver palette. After that, you use manager views to add child components: devices and proxy points This can be done even if the platform is offline.

Step 1 Open the driver palette that contains the network component.

Step 2 Expand **ConfigDrivers** in the Nav tree.

By convention, you should keep all driver networks under the station's **Drivers** container. It is a special component with a view that manages network devices.

Step 3 Do one of the following:

- Drag an instance of the network component from the palette to the **Drivers** container in the Nav tree.
- Copy an instance of the network component from the palette and paste it into the **Drivers** container in the Nav tree.

- Double-click the **Drivers** node, click the **New** button at the bottom of the **Driver Manager** view.

If you dragged or copied the component from the palette, a **Name** window opens. If you used the **New** button, the first of two **New** windows opens.

Step 4 Supply the information requested by the windows or accept the defaults and click **OK**.

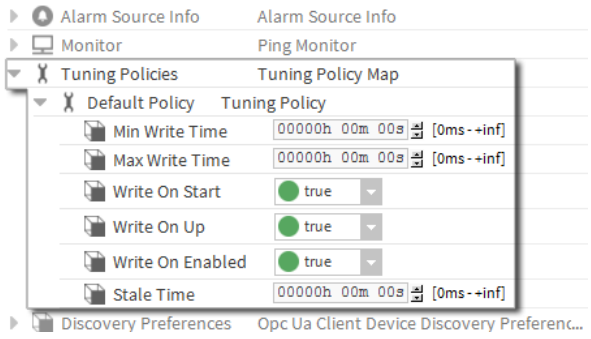
The Network appears as a row in the **Driver Manager** view

About tuning policies

A network's tuning policies configure rules for evaluating both write requests (for example, requests to write to proxy points) as well as the acceptable freshness of read requests that result from polling. Some drivers, such as BACnet, also support different poll frequency groups: Slow, Normal, and Fast. Tuning policies are important because they can affect the status of the driver's proxy points.

In a network's **Property Sheet**, expand the **Tuning Policies (Map)** slot to see one or more contained tuning policies. Expand a tuning policy to see its configuration properties.

Figure 7 Example Tuning Policies Map (OpcUa)



Some driver networks do not have tuning policies, for example, an **RdbmsNetwork**, and others, such as the **NiagaraNetwork** have simplified tuning policies.

By default, a driver's **Tuning Policy Map** contains a single default policy. However, you typically create multiple tuning policies, with different property values. Then, when you create proxy points under a network device, you assign a tuning policy to each point (as needed).

CAUTION:

Using only a single (default) tuning policy, particularly with all property values at defaults, can lead to possible issues in many scenarios. In general, you should create multiple tuning policies, configuring each based on the needs of the network's proxy points and the capabilities of the driver. In particular, tuning policy properties that specify writes from the framework should be understood and applied appropriately.

For example, under a **BacnetNetwork** you could change the default tuning policy's **`Write On Start`** property from the default (`true`) to `false`. Then, duplicate the default tuning policy three times, naming the first copy "Slow Policy", the second copy "Normal with Write Startup", and the third copy "Fast Policy".

In two of those copies, change the **`Poll Frequency`** property from `Normal` to `Slow` or `Fast`, corresponding to its name. In the "Normal with Write Startup" tuning policy, change its **`Write On Start`** property back to `true`.

The result would be four available (and different) tuning policies to apply when creating and editing proxy points.

Discovering objects

With the exception of drivers that do not support online discovery and the quick–learn option for discovering devices in a **LonNetwork**, object discovery (learn) is a two-step process: discover, then add or match. Objects to discover include stations, devices, points, schedules and histories. Discovery is available using the manager views. The managers provide a two-pane view with **Discovered** and **Database** tables as well as a single-pane view with only a **Database** table. This is a general procedure that works for all drivers.

Prerequisites: You are working in Workbench connected to any station.

Step 1 Navigate to a manager view, such as **Device Manager** (**NiagaraNetwork** node), **Points Manager** (**Points** node), or other node and double-click the node in the Nav tree.

The manager view opens.

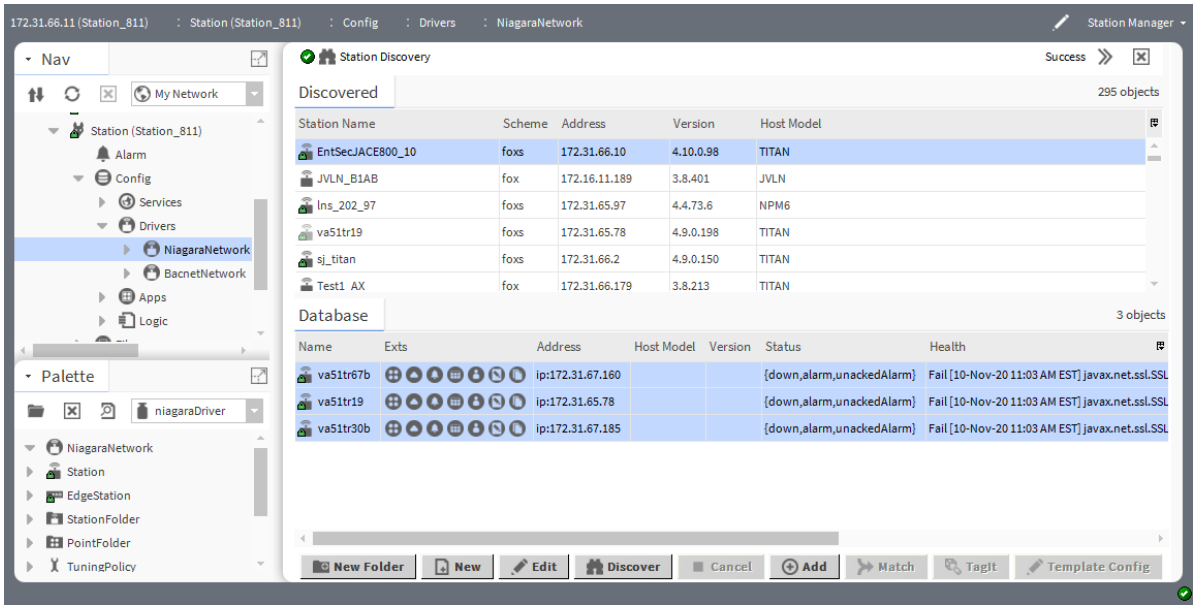
Step 2 Click the **Discover** button at the bottom of the view or the Discover tool in the toolbar ( ).

The Discover tool is available only if the driver supports online discovery.

Some drivers open an intermediate window before the discovery job begins to run. For example the BACnet driver opens a **Configure Device Discovery** window, which limits the discovery job to a specific range.

Step 3 Configure any discovery options and click **OK**.

The discovery job splits the screen into two panes and, as it runs, displays a progress bar along the top of the **Discovered** pane.



After the job completes:

- The **Discovered** pane lists the discovered objects that can be added to the database.
- The **Database** pane displays any objects already in the database.

You can drag the horizontal border between the panes up and down.

Step 4 Do one of the following:

- If a discovered object is not yet in the database, select it in the **Discovered** pane and click **Add** or click the Add tool ( ) or right-click the discovered row and click **Actions→Add**.

- If a record for the discovered object was added to the database while the station was offline, select the object in the **Discovered** pane and its database record in the **Database** pane and click **Match** or click the Match tool in the toolbar ( ).

  NOTE: Match is strictly a one-to-one function for discovered-to-database records. It is unavailable if multiple items are selected in either the top or bottom pane.

When you click **Add** or **Match**, a window opens. The properties in this window depend on the driver.

This window is nearly identical to the single-device **Edit** window and usually includes the object address. When matching, most properties already have acceptable values, otherwise communication to the discovered object would not have occurred.

Step 5 To edit an object database record, select it in the **Database** pane and click **Edit**.

Step 6 To toggle between learn mode (the view that displays both the **Discovered** and **Database** tables) and single-pane mode (**Database** table only), click the Learn Mode tool ( ).

Saving time when setting up identical objects

If you are setting up a network of many devices and points for the first time, you can speed configuration using the match feature. This procedure summarizes the steps involved in configuring multiples of the same type of object.

Prerequisites: You are running Workbench on a PC that is connected to your device network. All devices are connected to the network and available for discovery.

Step 1 Add the first device or point to the station database and configure its properties.

Step 2 For devices, add any Px views including all self-contained links and bindings.

Step 3 Duplicate the object by right-clicking it in the **Database** pane followed by clicking **Duplicate**.

The **Name** window opens.

Step 4 Give the duplicated object its own unique name.

Step 5 Repeat the duplication steps for all identical objects.

Step 6 Discover all objects.

Step 7 Use the match feature to associate each duplicated record in the station database with its discovered device or point.

This repopulates the necessary properties of the duplicated object with the correct values from the discovered object.

Step 8 Configure any unique properties for each object.

Programming offline

If you cannot connect your PC to the actual device network, you can manually add devices and points to the station database. Most components are available for configuration under a driver network. This procedure provides general steps.

Prerequisites: Your PC is running Workbench but is not connected to the device network.

Step 1 Set up your network component.

Step 2 Open the **Device Manager** by double-clicking the network component.

Step 3 Use the **New** button and window to add and configure devices.

Step 4 Add any unique point extensions.

Step 5 Open the **Point Manager** by double-clicking the **Points** folder.

Step 6 Use the **New** button and window to configure points.

Step 7 Use any other device extension managers to add objects or add saved applications (from the device level on down) to the station database.

Step 8 Once all devices, points and your PC are online, run a discovery job on devices and points.

Step 9 Use the match feature to associate the database records with the discovered devices and points.

Step 10 Edit any unique component and point properties as needed.

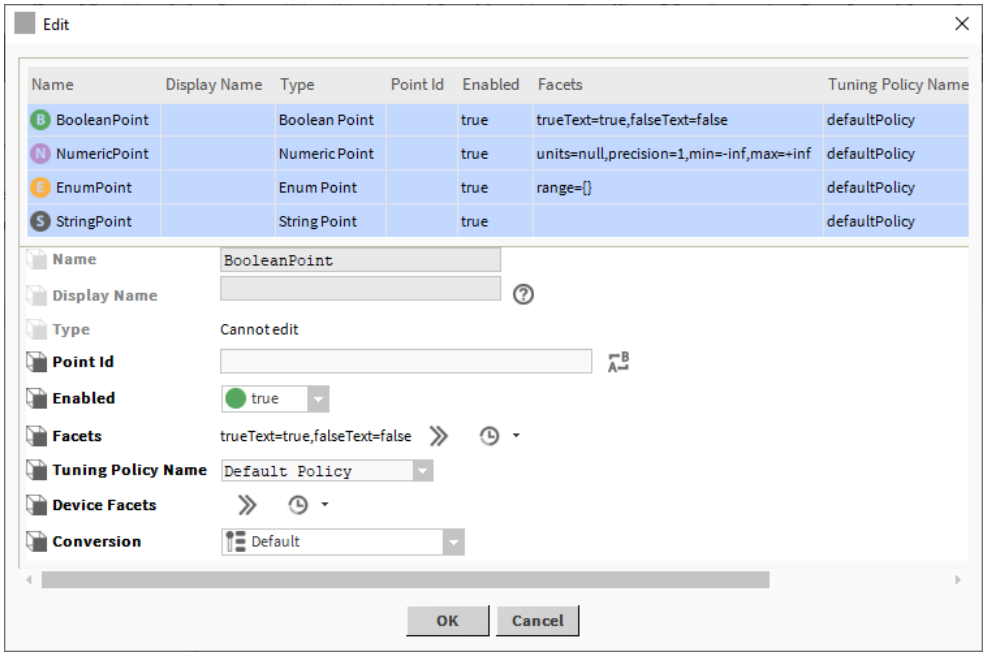
Group editing objects

Selecting multiple devices and point records in a station database saves time. Group editing makes the same change to all the selected points.

Prerequisites: You are using Workbench and are connected to a station (Supervisor or remote controller station).

Step 1 Access the manager (device or point).

Step 2 Use the standard Windows keyboard controls (**Shift** and **Ctrl** keys) to select multiple devices or points.



The screen capture shows four points ready to be edited at the same time. When you select multiple points, only properties that can be group edited are available. The others are dimmed.

Step 3 Make the change.

The change you make applies to all selected points. For example, you could change the tuning policy associated with multiple points at the same time rather than edit the tuning policy on each point individually.

Step 4 To edit a single object even if multiple objects were originally selected, select the single object and make your change.

Some properties that should be unique, such as address properties, may still be available to group edit as the rule about dimming out unavailable properties for group edit is not universally enforced. When editing properties, verify that only the point(s) you intend to edit is/are selected.

The change applies to only the specifically–selected objects.

Step 5 After making changes to all objects, click **OK**.

All changes apply to the objects.

Tagging objects

A tag adds additional information to a database record. You can tag remote station records in a Supervisor station database, device, point, and schedule records. Tagging is integrated in all of the manager views via Tag Mode and the **TagIt** button.

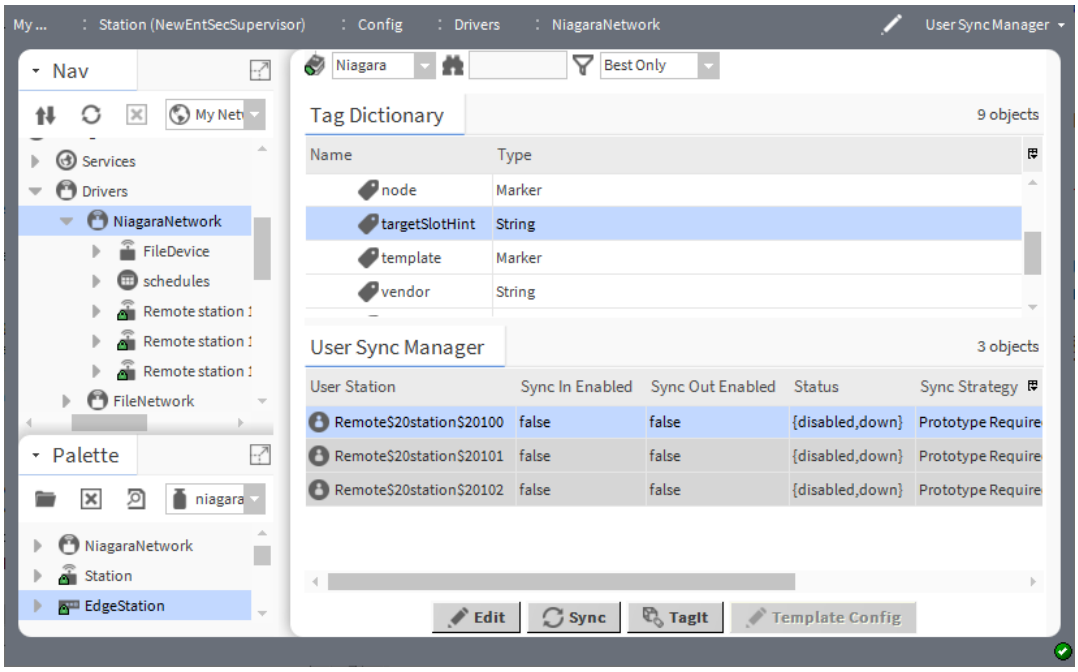
Prerequisites: You are working in Workbench connected to a Supervisor station.

Step 1 Navigate to the manager view that contains the objects to tag.

For devices, open the Device Manager by double-clicking the network component under **Config→-Drivers**. For points, double-click the station in the Nav tree and double-click the **Points** container.

Step 2 Select one or more records to tag and click the Tag Mode tool ( ) in the toolbar.

The **Tag Dictionary** opens either above the manager view or splits the top pane with another view and activates the **Tagit** button.



Step 3 Select the tag in the top pane and click **TagIt**.

The tag chooser opens.

Step 4 Configure the tag and click **OK**.

The **TagIt** button applies the selected tag(s) and displays the **Tag Added** confirmation window.

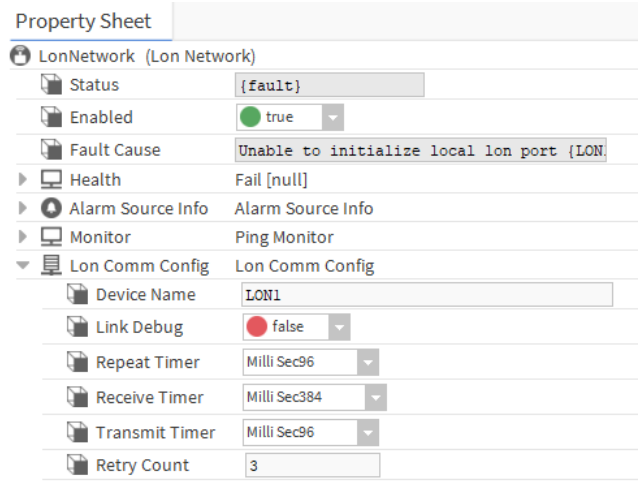
For more information on tagging objects, refer to the *Niagara Tagging Guide*.

About communication components

Communication components contain properties and possibly other components that configure and represent communications ports, or other protocol layers. These components vary from driver to driver. Often, you must configure (or verify) these components to allow online operations.

For example, a **BacnetNetwork** contains a BACnet stack (**Bacnet Comm**) and a **LonNetwork** contains **LonCommConfig**.

Figure 8 A LonNetwork's LonCommConfig component



See the driver guide for specific details on configuring a network's communications components.

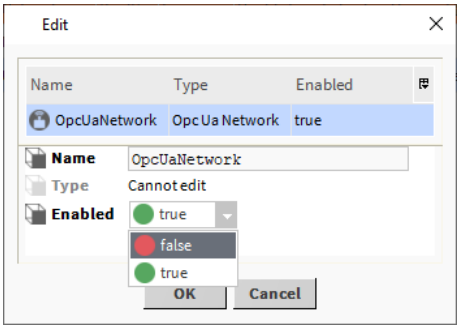
Driver troubleshooting

Each driver comes with its own challenges. This information applies to all drivers and may be a place to start.

Component disable

Disabling a component may be useful for maintenance purposes. To disable a network, select it in the **Driver Manager** view, click **Edit** and set **`Enabled`** to `false`.

Figure 9 Disabling the OpcUaNetwork



Disabling a network changes the status of all its child components (all devices, proxy points, etc.) to disabled. For a field bus driver, this action also suspends communications routines including polling and device status pings. By default, disabled components display in gray text on a light gray background.

You can also disable devices and proxy points. The same procedure applies: select the device or point in the **Device Manager** or **Point Manager**, respectively and click **Edit** and set **`Enabled`** to `false`.

A disabled device disables all child (proxy points) as well as point polling. Disabling a proxy point disables the single point.

Chapter 2 About device components

Topics covered in this chapter

- Device extensions
- Device state
- Device address properties
- About proxy points
- About polling

A device component is a second-tier component under a driver network component. Each device component has some number of required (frozen) slots and properties that are common to all drivers, and other slots, which vary by driver.

For example, a Lon device has a large collection of nv slots (network variables nvi, nvo, and nci). Each is an expandable container holding numerous properties. A BACnet device holds a number of properties that define the services it supports. For details, refer to the specific driver document.

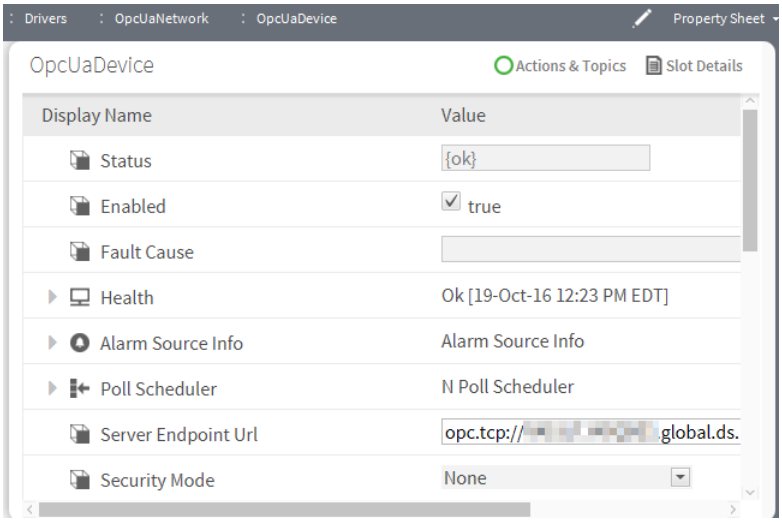
By default a device component has mandatory (frozen) component slots, such as Health and other status properties.

Depending on the driver type, a device typically has other properties. For example, if a device under a field bus, it may have a Device Address property, or a similar properties ir it may have a virtual gateway slot.

Each device has one or more device extensions, for example, a **Points** extension (DevicePointExt).

Typically, a **Property Sheet** is the default view for a device component.

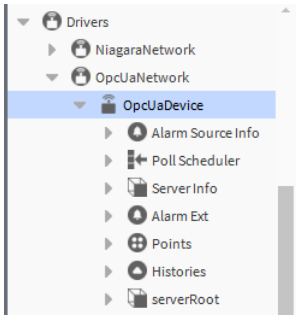
Figure 10 Example OpcUaDevice property sheet



You can access device properties, slots, and its extensions using the device's **Property Sheet**, as shown.

A device component is also the parent container for all device extensions.

Figure 11 Example BacnetDevice extensions in Nav tree



Device extensions, for example, **Points** are visible under the device when you expand it in the Nav tree.

Device extensions

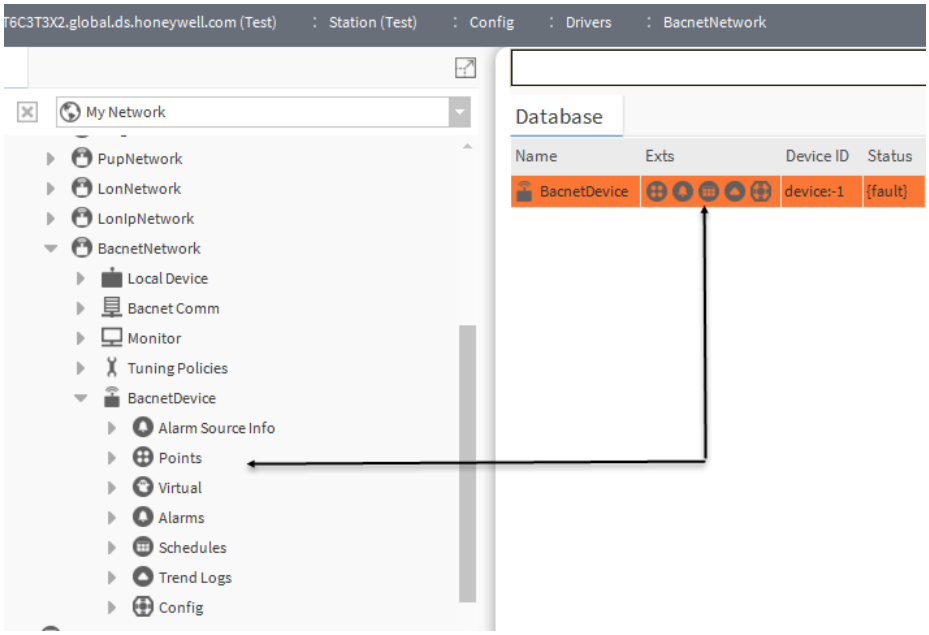
A device extension is a child container component of a device and represents some functionality of the device. Each extension contains properties and other components. Device extensions often have one or more special views.

For example, in any of the field bus networks, each device has a **Points** extension, which serves as the parent container for proxy points. The default view of the **Points** extension is the **Point Manager**, which you use to create and manage proxy points.

Device extensions are required (frozen) components of a device—you cannot delete them. Adding a device to the station database automatically creates them. This varies from the point extension model where you individually add and delete extensions under control points and components.

When you create a device-level component (New, Add, or drop a device from the driver's palette on a network node), device extensions group various functions of a device.

Figure 12 Device extensions in Nav tree and Device Manager



For any device, its extensions are typically visible both in the Nav tree and in the **Device Manager** view (as shown), providing double-click access to each extension's default view. A Supervisor's **Station Manager** view is an exception. In a **NiagaraNetwork**, special provisioning extensions for stations do not appear.

Common types of Device extensions include:

- Points extension: this is, perhaps, the most important of all device extensions. It serves as the container for all proxy points that represent real-time data originating from the device.
- Histories extension
- Alarms extension
- Schedule extension

The **NiagaraStation** component (the device in a **NiagaraNetwork**) has additional device extensions, including a **Users** extension, and **Files** extension and **Sys Def** extension.

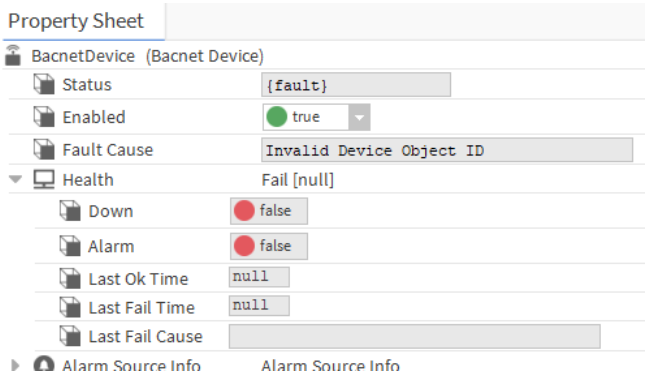
Device state

The common properties, **`Status`**, **`Enabled`** and **`Health`** provide information about the state of a device under the driver network. There are no hardware device components in the **`driver`** and **`niagaraDriver`** modules. This topic applies to all driver device components.

Status

The **`Status`** property for a device indicates whether it is communicating. A good status is `{ok}` (no status flags).

Figure 13 Device status properties



If you are using the **New** button to add a device, its initial **`Status`** may be `{down}` or `{fault}`, as shown here. This could mean that the device address properties are yet unassigned or are mis-configured.

NOTE: If a device reports a status of `{fault}`, see the **`Fault Cause`** property value for more details.

Successful polling or a device ping action verify the status of devices. You configure ping properties using the **Monitor** component.

From the Device Manager view, you can also right-click a device, and from the popup menu select **Actions→Ping** to manually verify communication with the device.

Depending on conditions, a device may have one of various status flags set, including fault or disabled or other flags in combination, such as down, alarm, stale, and/or unackedAlarm. In the **Device Manager**, non-ok status is indicated for any device by a row color other than white.

Enabled

By default, device **`Enabled`** is set to `true`. You can toggle this in the **Property Sheet** or in the **Device Manager** view (by selecting the device and using the **Edit** button).

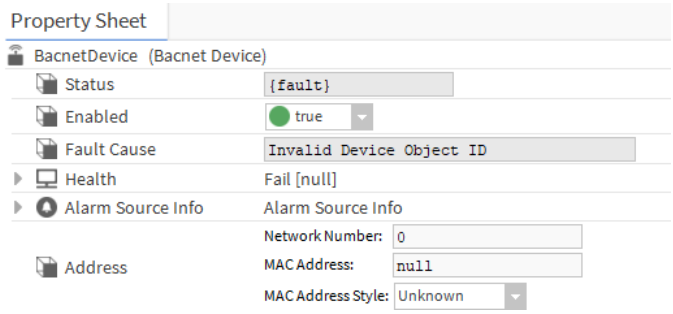
Health

This component contains historical properties that display the last successful message received from the device, including timestamps.

Device address properties

Depending on the driver, a device-level component typically has one or more address properties that are accessible from the device's **Property Sheet**. For example, a **BacnetDevice** has an **`Address`** property with three separate elements.

Figure 14 Example of BACnet device address properties



In the case of a Lon device (**DynamicDevice**) various address properties are stored under a **DeviceData** component, which is, itself, visible when you expand the device in the Nav tree. For more details on device-level address properties, see the specific driver guide.

About proxy points

In general, proxy points are any of eight control points, which often form the bulk of all points in a station. This is true for a controller or Supervisor station.

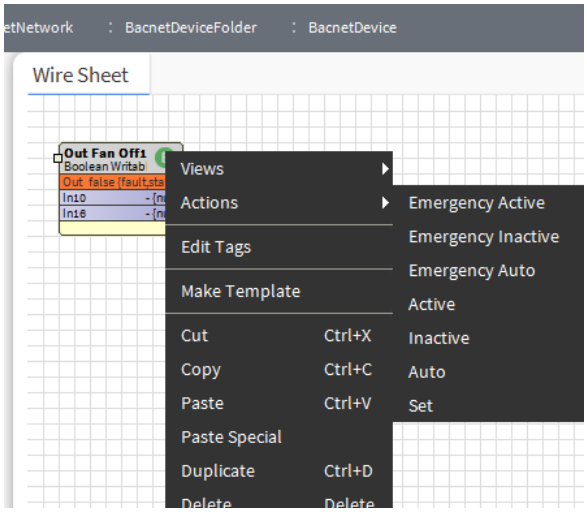
Point discovery

As with other objects, driver networks provide online "point learns"—online point discovery. This is true, for example with the **NiagaraNetwork**, **BacnetNetwork**, **LonNetwork**, and **NrioNetwork**. Whenever available, this method is the easiest way to accurately add proxy points to the station database.

Point actions

Proxy points are always read-only points: `BooleanPoint`, `EnumPoint`, `NumericPoint`, and `StringPoint`. No other types are available. However, if the source component is a writable point, the framework supports any actions (commands) for that component.

Figure 15 Actions on a writable BACnet proxy point



Writable proxy points include actions, such as in the example above.

Proxy of a proxy, other candidates

Often the remote source component of a proxy point is itself a proxy point. This is common under the **Points** extension of a device in a field bus driver network (Lonworks, BACnet, and so on) or if the remote station has its own I/O, in which case, the source component may be an Ndio proxy point.

Another possibility is for the remote source component to be a specific slot of a point extension under a point. For example, the **`Total`** property is such a slot under a **NumericTotalizerExt** component. In other cases, the remote source component may be a **`kitControl`** slot, such as one of the math or logic slots, or one of the other more complicated slots (objects): **LoopPoint** and **DegreeDays**.

Regardless of the source component, you model all remote data with proxy points by selecting one of the four, read-only point types.

About Points views

Although you initially use the **Point Manager** (default) view of a device's Points extension (and/or child point folders), typically other views are needed when engineering a network. You can use the view selector in the locator bar, or in the Nav tree right-click Points (and/or child point folders) and select one of the **View** menu options.

NOTE: Remember that the **Point Manager** view provides access only to proxy points, and edit features apply only to proxy extension properties.

Commonly used Points views include:

- **Wire sheet**

  Shows all proxy points, plus any kitControl and schedule components, simple control points, and so on, as well as links between them. Typically, you use this view to engineer control logic.

- **Property sheet**

  Also includes all proxy points, plus any kitControl and schedule components, simple control points, and so on. As needed, you can use this view to expand down to any level to access and edit properties. For example, you can access an alarm extension under a proxy point.

- **Slot sheet**

  Also includes all proxy points, plus any kitControl and schedule components, simple control points, and so on. As needed, you can use this view to edit config flags from defaults (say, for security schemes), edit config facets, and add name maps.

- **Px view** (**New** View)

  (Optional) A custom graphical view that you define by creating a new Px file or using an existing Px file. When created, this view becomes the default view for the device's Points extension (or if created for a points folder, its default view).

Proxy point best practices

The typical large scale use of proxy points has historically been in a Supervisor station to model and display real-time values centrally in Px views. Historically, a Supervisor station may have several hundred or even thousands of proxy points with data that originated within multiple remote stations.

Are proxy points even needed for Px views?

The introduction of export tags, metadata tagging and enhanced writable virtual points has diminished the need for proxy points in a Supervisor stations:

- If a Supervisor needs to serve up Px pages that already exist in a station, the use of PxViewTag export tags in that controller station can automatically replicate all needed components on the Supervisor, including all necessary virtual points. Proxy points are not needed.

- If you are engineering Px pages that exist only in the Supervisor station and you need real-time values from subordinate controller stations with access to right-click actions, consider using virtual points instead. In some cases, writable virtual points offer techniques previously harder to perform, such as providing a user edit access in a graphic to properties, such as alarm limits.
- Metadata tags associated with objects, including categories, roles (permissions), and hierarchies, provide access control and configuration options to manage automated buildings efficiently.

Avoid point extensions under proxy points

Although there is no rules to prevent you from adding history extensions or alarm extensions to proxy points, doing so may be unwise as even momentary communication issues between a controller station and its Supervisor station can result in loss of otherwise recorded history records and alarm conditions.

These recommendations apply to configuring roxy points:

- Instead of adding a history extension to a proxy point in the Supervisor station, add it to the source point in the remote station. Then, either import such histories into the Supervisor or, from the controller station side, export them to the Supervisor.
- Instead of adding an alarm extension to a proxy point in the Supervisor station, add it to the source point in the remote station. Then, configure alarm routing from the controller station into the Supervisor.

The routines for inter-station history archiving (import) and alarm routing provide integral retry mechanisms, such that temporary loss of station communication typically does not result in loss of history data or alarm events. Instead, that data safely reside in the controller until communication can be re-established.

When is a proxy point required

A proxy point is required when station control logic requires its data value, that is, as the source of a link. For example, you may have a local Average math component that is averaging temperature values sourced from three different stations, using proxy points. Virtual components cannot be used in this way because they do not persist. They exist only during active subscriptions (typically from users accessing Px pages).

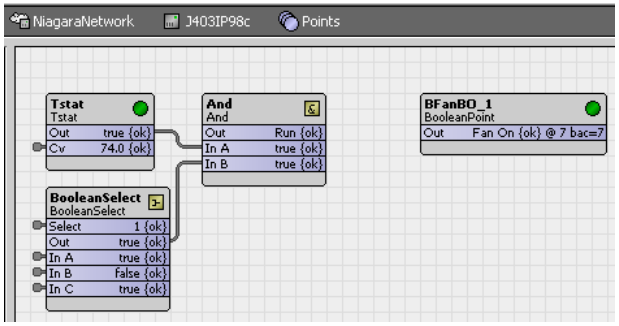
Controller stations that directly share and process selected data for control purposes require proxy points.

Link control and proxy points

As proxy points have no inputs, you cannot link into them from local station logic even if the remote source component is a writable point. To configure inter-station link control, make another proxy point in the remote station that proxies whatever local source component you need for the link.

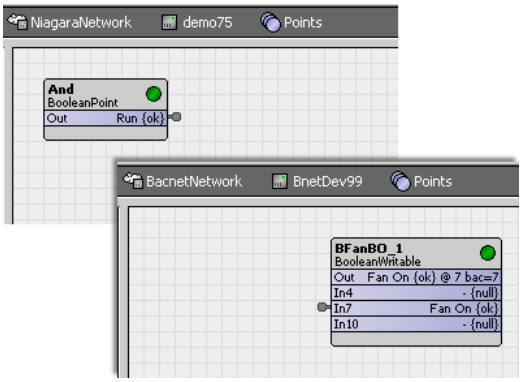
For example, consider a proxy point for a `BooleanWritable` that controls a fan. To provide link control from a local **And** object, which controls at priority level seven, your wire sheet might look like this:

Figure 16 Local object cannot link into a proxy point



In the example, you cannot link into the proxy point **BFanBO_1**. So, the remote station must make a proxy point for the **And** object, then link its output to the source point for **BFanBO_1**.

Figure 17 Source control object now a proxy point in the remote station



The example shows the wire sheet views with the source point and **And** objects.

In a typical Supervisor station that serves primarily Px views not much direct link control is needed so this rarely applies. In addition, the schedule import and export mechanism provides central scheduling control using local (imported) schedule components. However, if you are configuring applications between remote stations that require link control, always use proxy points in this fashion.

About polling

Polling updates device and proxy point values in a controller station database. Regardless of driver type, each polling service (named **Poll Service** or **Poll Scheduler**) operates in the same basic manner.

Many driver networks use a single polling mechanism and thread to adjust polling frequency at the network level in a station. The polling service samples device and point values at one of three poll rates: `Slow`, `Normal`, `Fast`. You configure the time interval associated with each rate. These are arbitrary names. There is no logic that enforces a relationship among their values.

As you work with each network you will encounter differences among pollable objects (pollables are objects that can be polled):

- A **BacnetNetwork** provides a separate **BacnetPoll** component for each port under its stack's network component using two threads per network port. You assign the **`Poll Frequency`** (poll rate) for a **BacnetNetwork** under its network **Tuning Policy**.

  NOTE: Multiple threads provide poll statistics that reflect the sum of activity for all the threads. There is no way to determine a statistics breakdown for each thread.

- Other drivers configure **`Poll Frequency`** for proxy points in each point's proxy extension.
- In the case of device objects, a **`Poll Frequency`** property selects the rate directly.
- The **NiagaraNetwork** does not provide device and point polling.

A **`Poll Frequency`** property is typically located below the device address properties. It provides a drop-down menu to select among the three poll rates as configured in the network's polling service.

In addition to the poll rates (`Slow`, `Normal`, `Fast`), a fourth group, `Dibs Stack`, supports pollables that transition to a subscribed state. This may be a temporary subscription, such as results when viewing unlinked proxy points (without alarm or history extensions) in Workbench or a browser. Or, it may occur when a polling service polls a permanently-subscribed point for the first time. Thereafter, it is no longer dibs-stack polled.

Every 10 seconds, the poll scheduler rebuilds the list of objects assigned to each poll rate. An algorithm breaks up each rate list into optimally-sized groups, which allows the poll thread to switch among the rates. This spreads the message traffic out evenly over the configured intervals causing points assigned to the quicker rates to update multiple times before points assigned to a slower rate update once.

Poll statistics are updated every 10 seconds. Fast, slow, and normal cycle times display the average time to complete a single poll cycle in milliseconds (ms). The poll scheduler algorithm automatically calculates an inter-message delay time to evenly spread out polling messages over the configured rate.

For example, if five points are assigned to a normal rate, the poll scheduler may poll a point in the list every two seconds resulting in a normal poll cycle time of around 10000 ms. This does not mean that it actually took 10 seconds to poll those five points.

Polling priority based on the assigned rate occurs like this:

1. The scheduler polls the dibs stack first. When first subscribed, a pollable moves to the top of the dibs stack (first dibs). The dibs stack is polled last-in, first-out (LIFO). As long as entries are in the dibs stack, the scheduler polls them as fast as possible with no delays.
2. When the dibs stack is empty, the scheduler uses the algorithm to poll the components in each rate group.

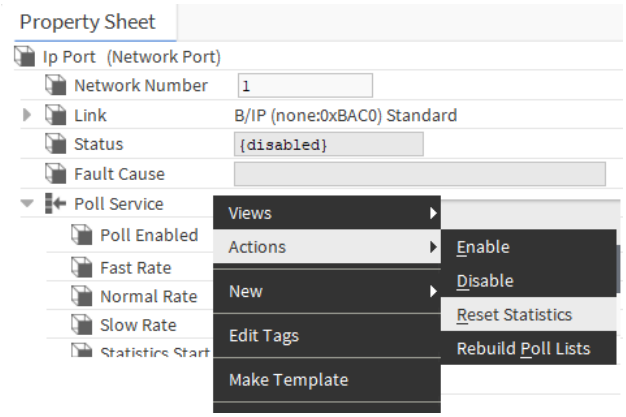
   NOTE: Every ten seconds the poll scheduler rechecks the rate groups for configuration changes. If a pollable's configuration changes from slow to fast, it takes at most ten seconds for the change to take effect.

Polling statistics

There are several statistics (polling service status properties) to watch when setting poll rate times and/or assigning points to different poll frequencies/tuning policies.

To manually reset poll statistics, right-click the Poll Service and click **Actions→Reset Statistics**.

Figure 18 Reset Statistics action of PollService



NOTE: Again, if a driver's polling service uses multiple threads (such as does the BACnet driver), the poll statistics reflect the sum of activity for all the threads—there is no way to determine a statistics breakdown for each thread.

How busy is too busy?—Poll rate tuning

A poll service with a busy time near 100% does not necessarily indicate a problem. It just means that the number of objects in the poll queues, along with the configured poll rate times are causing the poll thread to constantly execute instead of being able to periodically sleep. This means that the inter-message delay is basically at zero (0) seconds.

Some protocols, such as Modbus, use a silent time to define the start and end of a message. In this case, the inter-message delay is typically hard-coded as a minimum value. Other drivers, such as the American Auto Matrix PUP driver, include a property to define the inter-message delay time. In this case, the polling service use the inter-message setting as its minimum. In both of these cases, the inter-message delay may be longer than the configured minimum, but it will never be less than the minimum time.

When points are first subscribed, the poll scheduler adds them to the dibs stack for immediate polling. After this initial poll, it moves them to their assigned polling rate group, which results in a longer poll cycle time for the group.

Typically, a network has a base number of points that are permanently subscribed, including any points that are linked, and/or that have a history or alarm extension. These permanently-subscribed points always impact their assigned poll rate group queue. A good starting point would be to adjust the configured poll rates such that the busy time is at 75% when the scheduler polls only these points.

Maintaining some free time for the polling service should allow subscription surges (when users view graphics, property sheets, wire sheets, etc.), as new points are processed, without causing a noticeable increase in the poll cycle times. If poll cycle times do change noticeably during normal operations, you may need to adjust the poll rates such that a steady-state busy time is below 75%.

Assuming that the poll cycle time for a rate group is approximately equal to the configured rate, and the busy time does not indicate that the poll scheduler is constantly polling, it should be possible to reduce that configured rate value and speed up polling. This should result in quicker poll cycle times, which would be confirmed by seeing the rate group's actual poll cycle time decreased to something near the newly configured value along with an increase in the poll service's busy time.

If a group's actual poll cycle time is longer than the configured time, the objects in the poll list are being polled as quickly as the driver can. Decreasing the poll rate value (to speed up polling) has no negative effect on the busy time, but it will not speed up the polling process either—unless you reduce the number of points in the queue.

If a controller's CPU usage is high, and the poll service's busy time is near 100%, increasing poll interval rate values (to slow down polling) should reduce both the busy time and the CPU usage.

NOTE: Some drivers use a combination of polling and other mechanisms to update point values, such as the **LonNetwork**, which uses **nvUpdate** rules for bound links. A polling service's busy and poll cycle times do not reflect any statistics regarding points that update values using these other methods or any similar mechanisms.

Chapter 3 Getting started with the NiagaraNetwork

Topics covered in this chapter

- ♦ Core driver modules and palettes
- ♦ Adding a station to the NiagaraNetwork
- ♦ Alarm strategy
- ♦ About schedules
- ♦ Firewall changes to support discovery (foxs)
- ♦ About persisting fetched tags
- ♦ Adding more persisting tags
- ♦ History import and export
- ♦ Network users
- ♦ About virtual component spaces
- ♦ Setting up the Station Manager to display index status

The **NiagaraNetwork** models real-time data values from subordinate stations as proxy points, the lowest tier of components in a driver's hierarchy. Depending on the driver and devices, the **NiagaraNetwork** also integrates schedules, alarms, and data logs (histories).

To simplify driver modeling, the **New Station** wizard automatically creates the necessary **Drivers** container complete with a **NiagaraNetwork** and its required component slots.

Core driver modules and palettes

Each driver has one or more modules associated with it. All Niagara stations include the **NiagaraNetwork**, with general-purpose modules and palettes.

Table 1 NiagaraNetwork modules

Module	Description
driver	Supports stations with the **Drivers** container (network-level container), file management components, history components, ping monitor, timers and tuning policies.
fox	Provides the **FoxService**, which resides in the **Services** folder under **Config**.
niagaraDriver	Provides components that configure the connection between the remote controller station and Supervisor stations. These components reside in the **Station** folder under the **NiagaraNetwork**.
niagaraVirtual	Supports the **NiagaraNetwork**'s virtual components.
serial	Configures the serial port of any serial-based driver (Serial Port Config).
tunnel	Supports remote client access to local devices.

Table 2 Palettes

Module	Description
driver	Provides **TuningPolicy** and file-related components. Tuning policy properties control write times and stale time. File components and properties configure the **NiagaraNetwork** to receive data from external files.
fox	Provides components for configuring Fox and Foxs ports, and to manage server connections.

Module	Description
niagaraDriver	Provides components to configure the **NiagaraNetwork**.
tunnel	Provides the components for serial tunneling.

Adding a station to the NiagaraNetwork

The devices (stations) under a station's **NiagaraNetwork** can include upstream stations, peer stations and downstream remote controller stations. Using the **NiagaraNetwork**, the central station (usually a Supervisor station) connects to each station for the purpose of collecting configuration and update data. Any station (regardless of platform type) can be a client or server based on the **NiagaraNetwork** configurations on either side. Typically, Supervisor stations are the main client of downstream stations, but you could also set up a two-way connection, or even have a controller station be the client to the Supervisor.

Prerequisites: You are working in the client station (usually your Supervisor station) using Workbench. One or more remote stations share the same network.

Step 1 Expand **Config→Drivers** and double-click **NiagaraNetwork**.

The **Station Manager** opens.

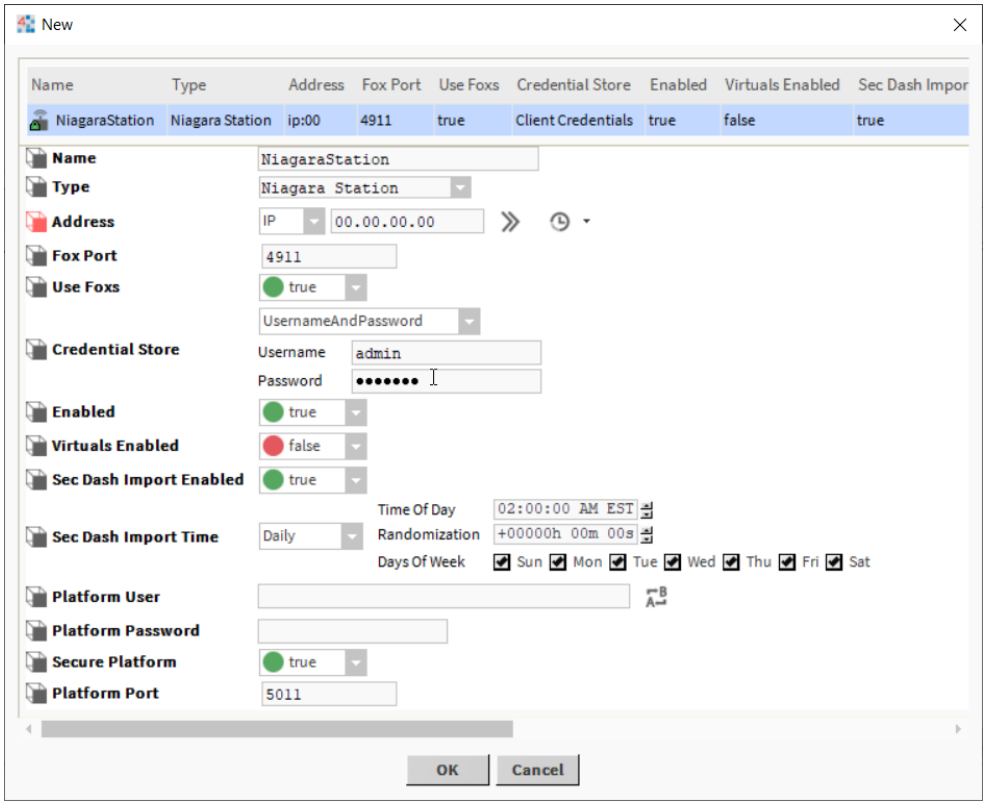
Step 2 To discover the server stations, click **Discover**, select the discovered station(s) and click **Add**.

Step 3 To manually add a server station, click **New**.

The **New** window opens.

Step 4 Select the station type, the number of stations you want to add, and click **OK**.

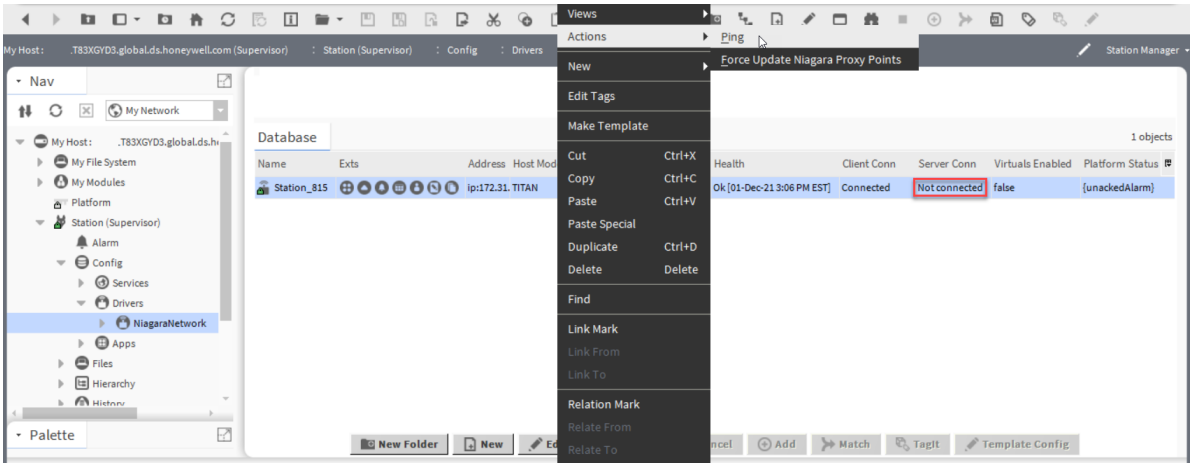
The **New** window opens.



Step 5 As a minimum, enter these properties and click **OK**:

- **Name** is the name of the server station. Each **NiagaraStation** under your **NiagaraNetwork** requires a unique name. If you are not sure of this name, connect to the station. The **Station Name** is listed under **Summary Properties**.
- **Address** is the IP address or host name of the server station.
- **Credential Store** (**Username** and **Password**) are the user credentials required to access the server station.

The **Station Manager** lists the newly-added station in the **Database** pane.



Step 6 To debug the connection, do one or both of the following:

- If the Server Conn column reports `Not Connected`, right-click the station row in the **Station Manager** and click **Actions→Ping**. This pings the client to establish the connection with the server.
- If the Status column reports `down`, right-click the station row and click **Views→AX Property Sheet**

The server station's **Property Sheet** opens.

Step 7 If **Health** reports `Fail`, expand **Health** and review the **Last Fail Cause**.

A **Last Ok Time** of `null` indicates that the station has not connected even once. Several errors may be preventing the client station from connecting to the server station:

- `Permission denied: connect` is a general error. Edit the configuration and/or ensure that the server station is available to fix the problem.
- `Station name is same as this station` means that you attempted to make an illegal loopback connection (the same station attempting to connect back to itself).
- `certificate_unknown` means that the station (client) cannot authenticate the self-signed certificate sent to it from the remote station (server).

  If you trust the certificate, expand **Services→PlatformServices**, double-click **CertManagerService**, click the **Allowed Hosts** tab, select the certificate whose host name is the same as the IP address (or host name) of the server station (its icon should be a white X on a red shield), click **Approve** and click **Yes**. The red shield changes to a white check mark on a green shield.

  Approving this self-signed certificate should be a temporary measure. Each remote server station/platform should have its own server certificate that has been signed by a certificate authority. This enables server authentication, which is the best security practice for secure device-to-device communication.

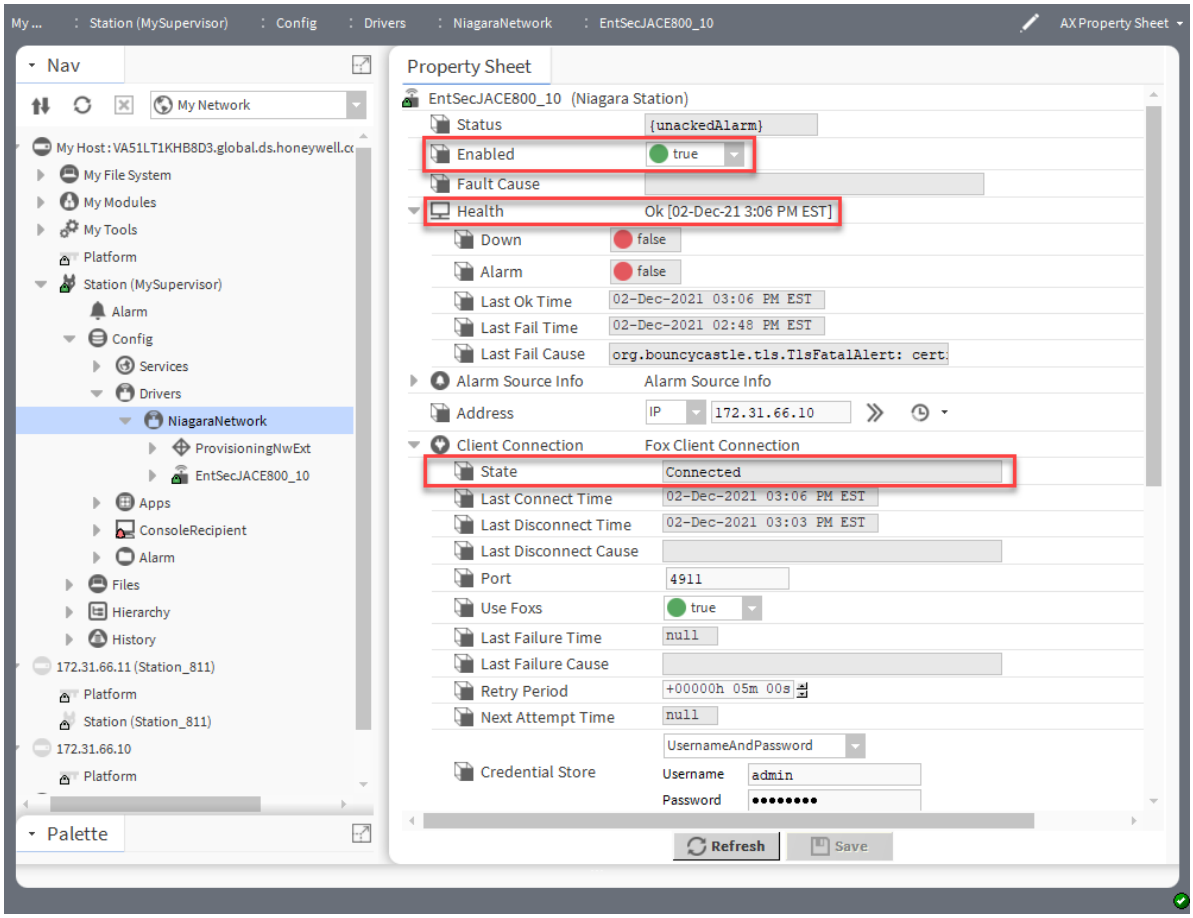
If the **Last Ok Time** is older than the **Last Fail Time**, something could be wrong or the station connection could be temporarily down.

Step 8 After correcting a problem, return to the **NiagaraNetwork**'s **Station Manager** under your client station and ping the station again.

The **Station Manager** should report that the station is `Connected`.

Step 9 To confirm the connection, right-click the station in the Nav tree, click **Views→AX Property Sheet** and expand **Health** and/or **Client Connection**.

The station's **Property Sheet** opens.



`Enabled` defaults to `true`. If set to `false`, **`Status`** reports `disabled`.

Health reports `Ok`. Notice that the **`Last Fail Cause`** reports the error that caused the last failure. You can ignore this error message as long as **Health** reports `Ok`.

The **`State`** property under **Client Connection** reports `Connected`.

Alarm strategy

Alarms generated by a driver in a remote controller station can be configured to display in the Supervisor station.

To configure alarms in one station to be received in another station you add a **StationRecipient** under the **AlarmService** container of the sending (source) station. You then link the **AlarmClass** component to the **StationRecipient**.

It is not necessary to use the same **AlarmClass** component in the both stations (although that is one approach). In the receiving station (often, the Supervisor), you can configure all alarms from a remote station to route to a single local **AlarmClass** or, use a prepend or append scheme to route to different (but associated) alarm classes, where all schemes work based on the names of the alarm classes.

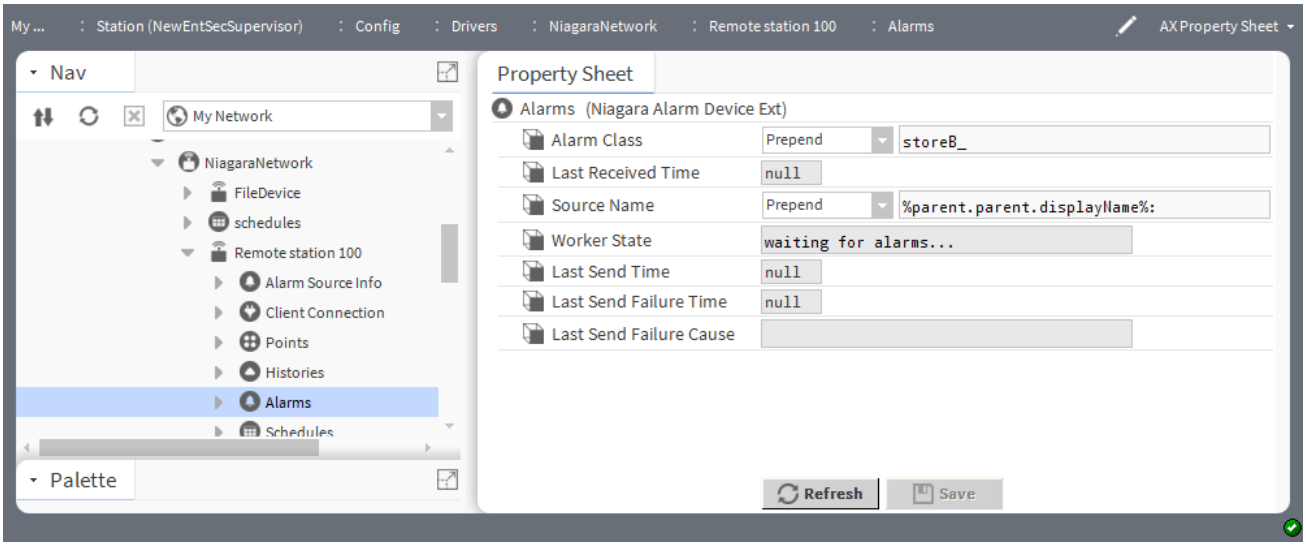
Prepend and append

The **`Alarm Class`** property of a **NiagaraStation**'s **Alarms** extension offers two options: `Prepend` and `Append`. An associated text string adds a message. If you set up each remote station under the Supervisor's **NiagaraNetwork** to replicate the station database, each remote controller station can have identically-named **AlarmClass** components along with other identical characteristics.

Consider a large chain store where each remote controller is installed at one retail store and each controller station these alarm classes: tempAlarm, humAlarm, runAlarm, accessAlarm, and so on. In the Supervisor station, you want separate alarm classes (and routing) for individual stores, so you create **AlarmClass** components named storeA_tempAlarm, storeB_tempAlarm, and so forth.

Now in the Supervisor's **NiagaraNetwork**, under each **NiagaraStation**'s **Alarms** extension, you would set the **Alarm Class** property to pre-append the store identifier to the alarm class name: storeA_, storeB_, etc.) as you set up when you created and named the **AlarmClass** components in each remote station.

Figure 19 Alarms properties under a NiagaraStation



When a store alarm comes to this Supervisor, the station adds the prepend text to the originating **AlarmClass** from the remote station such that routing looks for that named **AlarmClass**, for example: storeB_humAlarm or storeW_runAlarm. This maintains the original alarm class mapping at the store station level as well as segregating by store in the Supervisor station.

NOTE: This strategy does not automatically create virtual alarm classes in the Supervisor station. You still have to manually create all the needed **AlarmClass** components. **AlarmClass** component naming should match the Prepend or Append scheme as configured under the **NiagaraStation**s in the Supervisor station's **NiagaraNetwork**.

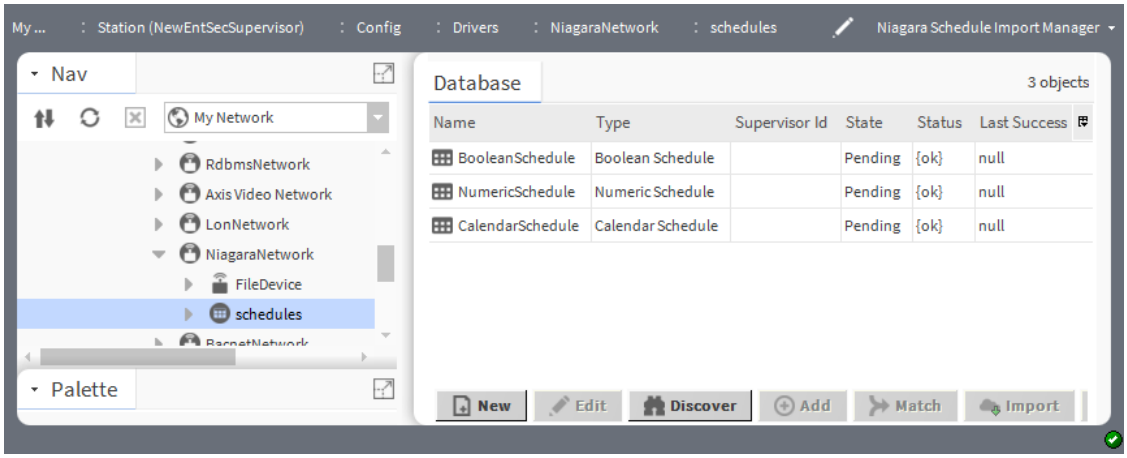
About schedules

Schedules control the activities of other components. Schedules originate in a source station, Supervisor or remote controller and can be transferred to another station.

Schedules under the NiagaraNetwork

Under the **NiagaraNetwork** of a Supervisor or controller station, a schedule device extension (**NiagaraScheduleDeviceExt**) serves as a container component for schedules (Boolean, numeric, calendar, etc. **NiagaraScheduleImportExt**s) imported from another station.

Figure 20 Example of the Schedule Import Manager under the NiagaraNetwork



The buttons at the bottom of the **Schedule Import Manager** create and configure the sharing of schedule components among stations. The receiving station manages this transfer. Unless you are working offline, you can use the learn mode (object discovery) in the receiving station to import schedules.

After it imports each schedule component, the software automatically creates a corresponding schedule export descriptor in the sending station. A separate **Export Manager** view provides access to schedule export descriptors. If necessary, you can review and adjust the export descriptor properties using the **Schedule Export Edit** window.

NiagaraStation schedules

A Supervisor station has an additional use for the schedule extensions. Under each of its **NiagaraStation**s, a **Schedule** device extension contains the import schedule extensions found in each remote station. The manager views under each **NiagaraStation** work uniquely from the way they work in other drivers.

By default, when you import a schedule under a **NiagaraStation** using the **Schedule Import Manager**, the import/export setup is initially configured on both sides as follows:

- On the receiving (slave) side, the driver configures each schedule import extension component with an **`Execution Time`** of `Manual`. Creating a schedule imports (pulls) the source schedule's configuration only once. You can manually import it again.
- On the sending (master) side, a corresponding export descriptor configures the schedule transfer with an **`Execution Time`** of `Interval` set to five (5) minutes. You can adjust this value using the **Schedule Export Edit** window.

  To review all properties of a schedule export descriptor, including **`Status`**, use its **Property Sheet** in the Nav tree under **Schedules** by double-clicking on the Export tool ( ).

  The default configuration does not mean that the driver continuously exports (pushes) the same schedule configuration to the remote schedule at the interval rate. Instead, the export descriptor keeps a **`Subordinate Version`** timestamp from the last export. If a configuration change occurs, the export descriptor compares this time against the configured interval, and, if necessary, exports the schedule to the remote station.

Reverse push

The default master push configuration is the most efficient (and recommended) way to keep imported schedules synchronized. However, if the network does not allow this method (perhaps due to firewall restrictions), you can configure things in reverse: set up the receiving sides **`Execution Time`** for `Interval` and disable the corresponding schedule export descriptors. This periodically re-pulls the schedule configuration.

Firewall changes to support discovery (foxs)

To successfully discover stations under a **NiagaraNetwork**, a Supervisor PC's Windows firewall needs exceptions for the PC's UDP port(s) as well as for its TCP port(s). This is because station discovery uses UDP multicasting. Otherwise, with only a TCP port opened, your discovery may come up empty.

These foxs ports require firewall configuration:

- TCP port 4911 (1911 if using fox and not foxs)
- UDP port 4911 (1911 if using fox and not foxs)

NOTE: Platform communication does not use UDP multicasting. Use of fox instead of foxs is not recommended because of the security risk.

If a Supervisor uses the standard port for platform connections (5011 for a secure connection, 3011 for a connection that is not secure), you might also add an exception for this port or for whatever TCP port the platform daemon uses.

If you are using non-standard port numbers for station (foxs) connections, similar firewall exceptions may be needed for those TCP and UDP ports.

About persisting fetched tags

In Niagara, there is an added frozen Boolean property on the **NiagaraNetwork** called **`Persist Fetched Tags`**. When set to `true` for any descendent station proxy point, it causes the `n:history` tag (and any other tags configured for fetching) to persist on the proxy control point as read-only metadata (as a direct tag).

This is useful in conjunction with the Niagara feature that allows you to add a proxy point to a web chart I a Supervisor station. The added proxy point resolves the local history to initialize the web chart. If the connection to the remote station goes down, persisting the `n:history` tag finds the local history (provided you previously visited the point when the connection was up) and persists the `n:history` tag information.

NOTE: The `n:history` tag is an implied tag that is also added to local control points if they have an active history extension child. The tag is not limited only to proxy points on a Supervisor station.

You enable the **`Persist Fetched Tags`** property on the client side of the NiagaraNetwork connection. That is to say, whichever is the consumer of the data. Typically, the Supervisor station is the enabled client since it would be fetching from the remote controller. But in a controller-to-controller connection you enable the property on whichever of those clients is the consumer of the data.

Related properties

You may add two optional properties that specify additional remote tags to pull from the remote controller to Supervisor and persist on the corresponding proxy points (or virtual components). The properties are: **`tagsToFetch`** and **`persistVirtualFetchedTags`** (`false` by default). These optional dynamic properties are only useful when the **`Persist Fetched Tags`** property set to `true`. To configure either of these optional properties, you must first add the slot to the client side proxy points, and then in the **Property Sheet** view enter the comma delimited names of additional tags to be fetched.

NOTE: When the **`Persist Fetched Tags`** property is enabled by default, the driver fetches the `n:history` tag regardless of the value of these optional properties.

Adding more persisting tags

You may add two optional properties to the **NiagaraNetwork Property Sheet**. These two properties specify additional remote tags to pull from the remote station to the Supervisor and persist on the corresponding proxy points (or virtual components).

Prerequisites: You are working in Workbench and are connected to the client side station (usually your Supervisor station). Each remote station has one or more tags assigned to its components.

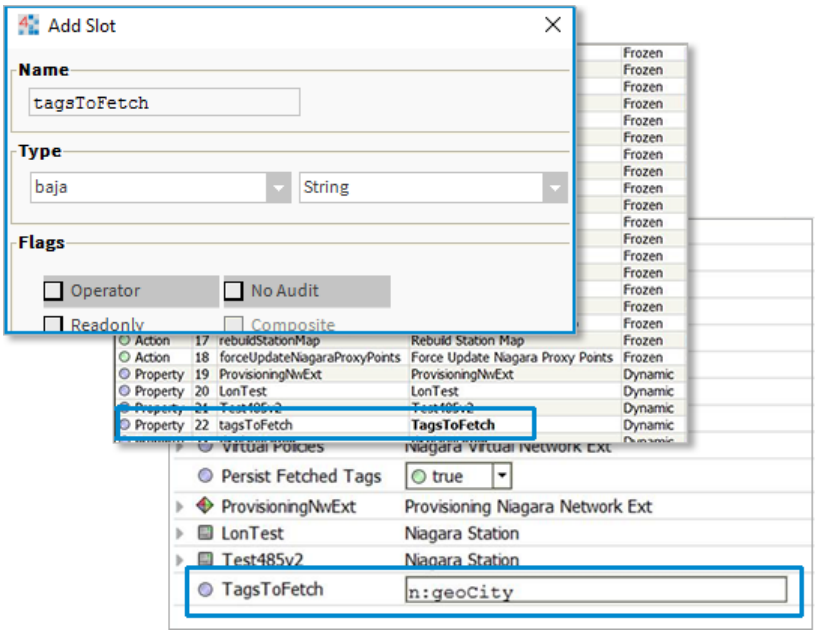
NOTE: For any specified tag to be successfully fetched from the server-side station to the client-side station, the tag must be assigned to components on the server-side station.

Step 1 Expand **Config→Drivers**, right-click the **NiagaraNetwork** and click **Views→AX Slot Sheet**

The **Slot Sheet** opens.

Step 2 To add slots, click **Slot Sheet→Add Slot** (menu).

The **Add Slot** window opens.



Step 3 Give the slot the name **`tagsToFetch`** with the **`Type`** of `baja:String`, and click **OK**.

This creates a dynamic property.

Step 4 Create a second dynamic property with the name **`persistVirtualFetchedTags`** and **`Type`** of `baja:Boolean`.

Step 5 Back on the **Property Sheet**, enter the full tag for each tag to fetch using the syntax: `tagDictionary:tagName`.

The tags may be on proxy points and virtuals.

CAUTION: Fetching tags that override valid implied tags on the Supervisor proxy point can cause confusion and unintended results. For example, it would not make sense to fetch certain implied tags such as the `n:type` tag because that would override the appropriate `n:type` tag already implied on the Supervisor proxy point.

Step 6 Set **`persistVirtualFetchedTags`** to `true`.

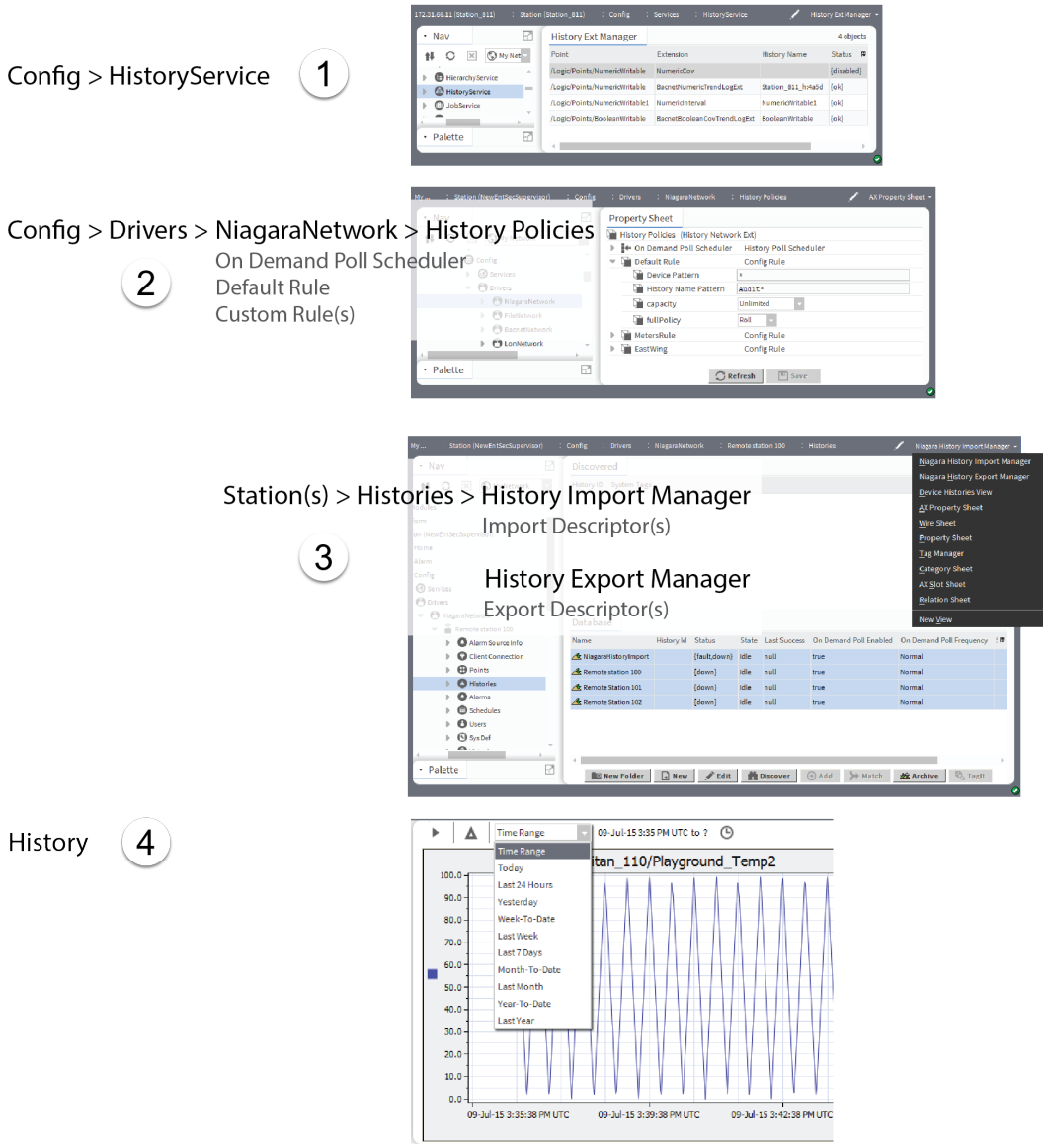
This property defaults to `false`. When set to `true`, the driver adds any fetched tags under the level of this property's container as properties (read-only) on the virtuals. Essentially, this means they are stored as direct tags on the virtuals.

History import and export

A history is a collection of time-stamped records that provide device point and system event values. The **HistoryService** manages histories in each station. The **NiagaraNetwork** manages the import and export of histories between stations. The *Niagara Histories Guide* provides pocedures for managing histories in a station.

Each station can both export and import history data. In practice, data are collected locally in remote controller stations. Connected to a remote controller station, you can export those history data to a Supervisor station (push) or, connected to a Supervisor station, import data from the remote controller station (pull).

Figure 21 Station history components in a target station



The graphic identifies all the station components that relate to histories. The component paths are designed to imitate the location of each component in a Nav tree.

1. The **HistoryService** manages the history database in the local station. The local station is the station you are currently connected to. This table view has a row for each local history.
2. **History Policies** (**History Network Ext** on the **NiagaraNetwork Property Sheet**) is a container for the **On Demand Poll Scheduler**, the default configuration rule and other configuration rules.

   The poll scheduler sets up polling frequency and reports polling status for imported histories. Usually these are histories transferred from a remote station to the Supervisor station.

   History policies select which histories to export from a remote station to a target station. Unlike imported histories, which let you define and adjust capacity and full policy properties later, if needed, the histories a station exports to another station have no associated component—only the history itself. The target

station's history policies set the capacity and full policy properties at creation time for each exported history.

3. The **History Import Manager** contains an import descriptor for each type of history. Each descriptor defines the name and ID of the history, when to pull it from the source station and how to manage the target database. You use the **History Import Manager** when you are connected to the Supervisor station and wish to pull data from one or more remote stations.

   The **History Export Manager** (selected from the drop-down list) contains an export descriptor for each type of history. Each export descriptor defines the name and ID of the history and when to push it from the source (local station) to the target station. You use the **History Export Manager** when you are connected to the remote station and intend to push data to the Supervisor station.

4. The **History** space (**history-FoxHistorySpace**) charts the histories in the local station.

Archiving

History import and export descriptors save a history to a different location (station) from where it originated. In a typical application, this is considered archiving. For example, in the originating controller station, a history may contain a limited number of records. In tSupervisorhe Supervisor station the same history may have unlimited record capacity. This topic provides information that is unique to histories under a **NiagaraNetwork**.

History features:

- Upon a join command, export tags in a controller station automatically add pre-configured history import descriptors to the **NiagaraStation** under the Supervisor station's **NiagaraNetwork** and import the histories from the controller station to the Supervisor station. This is one of many functions provided by export tags.
- The **Niagara History Import Manager** and **History Export Manager** view support folders under the to better organize history import and export descriptors. Each folder provides these manager views.
- The **Device Histories View** on a **NiagaraStation Histories** extension provides shortcuts to histories. Although not unique to the **NiagaraNetwork**, this feature may be useful, especially in a Supervisor station.
- On-demand polling for both local histories and imported histories makes it possible to poll for live data when viewing a history chart or history table view. The **Live Updates** button (play icon) toggles this feature.
- You can import and export histories based upon system tag text patterns rather than by using explicit **`History Id`**s, that is, using a discovery job in the **History Import (or Export) Manager** you can select specific histories to add. This utilizes a System Tags property in the history extensions.

The last two features above have associated properties in history import and history export descriptors with values available in the corresponding manager views. In addition, the on-demand polling feature has an associated **On Demand Poll Scheduler** component under the **NiagaraNetwork**'s history network extension (**History Policies**).

Rules for two exporting stations

This example uses two remote stations (WestWing403 and EastWing403) to illustrate history policies.

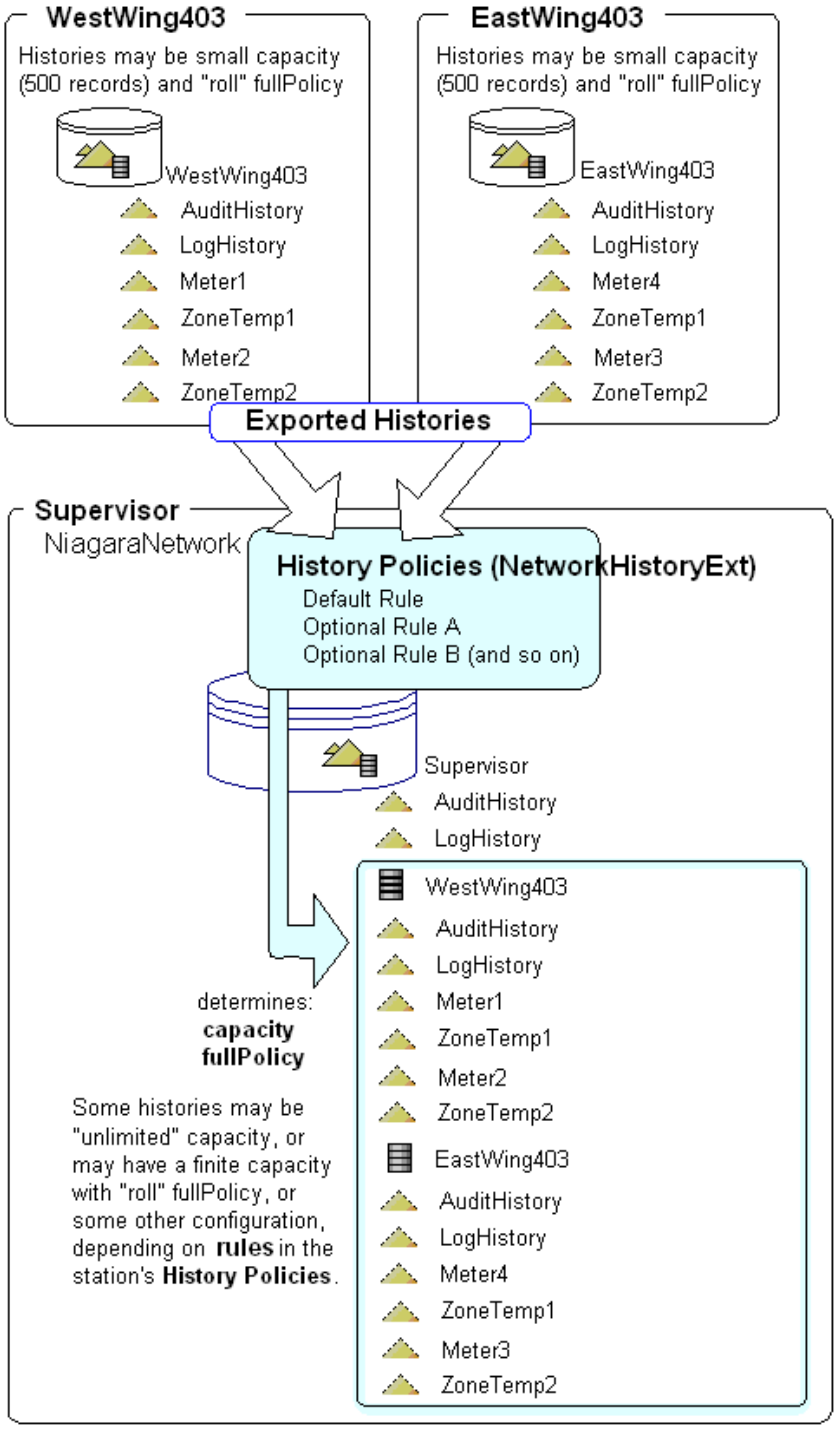
You export histories from a source (usually a remote) station to a target (usually a Supervisor) station working under the source station's **Histories** device extension of the **NiagaraStation** that represents this (local) station. Before exporting histories you need to set up one or more configuration rules in the target station for each source history. A configuration rule defines these properties:

- Two pattern properties identify the source station and its history names. These properties provide wildcards for selecting multiple stations and histories.
- The rule includes two properties: **`capacity`** and **`fullPolicy`**, which configure how the target station (usually the Supervisor) manages the histories.

Given the storage capacity of a Supervisor PC, a single, default rule may be all you need. If, for some reason, you are exporting histories to a remote controller station, you should definitely change the default configuration rule to specify smaller capacities. Even for a Supervisor station, you may need to change the default rule, and/or add additional optional configuration rules.

When a station exports a history, the target station uses the first matching rule to evaluate the data, and set up the capacity and full policy for the local (archived) history. The default rule is always at the top, and cannot be deleted or renamed. Other rules follow the default rule.

Figure 22 History Policies and configuration rules

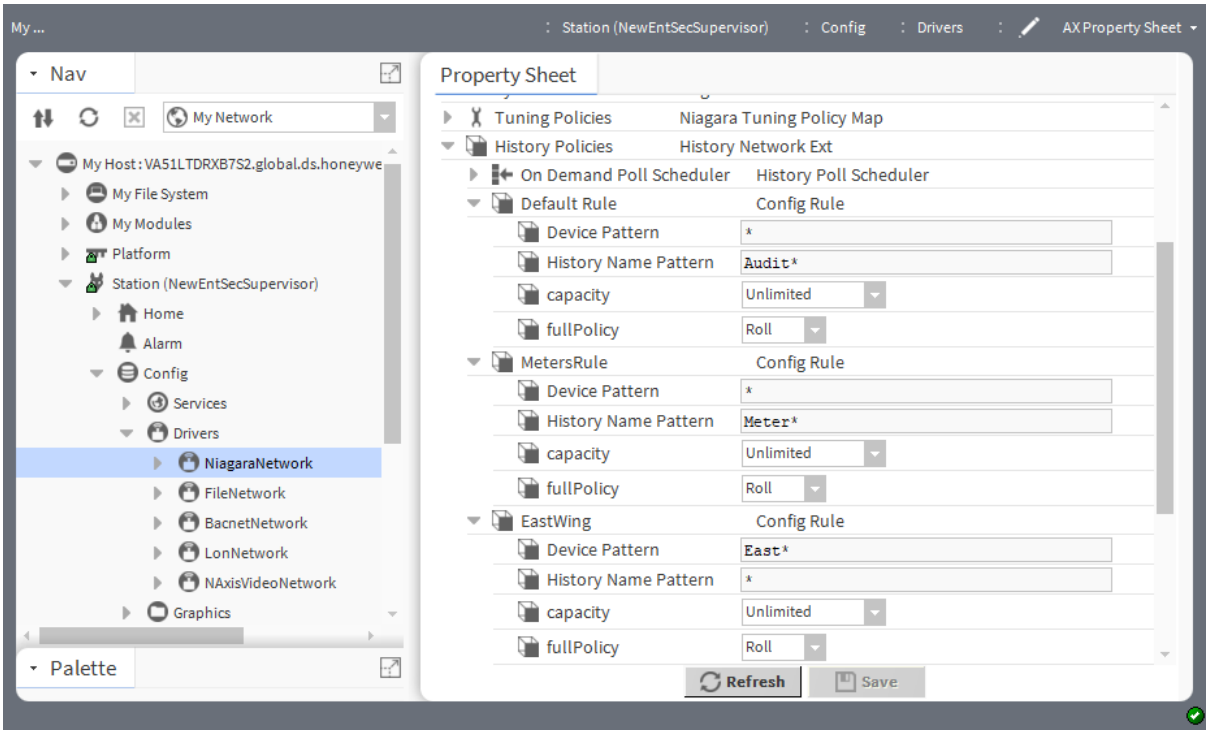


The Supervisor's history policies configuration rules determine the capacity and full policy settings for each history at the time the station exports them from WestWing403 and EastWing403. Rules are matched to remote station (device) names and history names, which determine the corresponding capacity and full policy values to apply upon history creation. If a history does not match any rule, the system creates it using the capacity and full policy defined for the source history.

Multiple configuration rules

In this example, the target Supervisor station has two additional configuration rules under the **History Policies** container of its **NiagaraNetwork**.

Figure 23 Three configuration rules



The highest-priority **Default Rule** matches all exporting stations with a history name pattern of `Audit*`. This wildcard would match any `AuditHistoryLog`. **`Capacity`** is set to `unlimited`, which exports all audit history records for inclusion in the archive.

The **MetersRule** configuration rule applies to all stations, and all histories with names that begin with Meter (`Meter*`). **`Capacity`** is set to `unlimited`, which exports all such histories for inclusion in the archive.

The **EastWing** configuration rule applies only to stations whose names begin with East (`East*`) and include all histories in the archive. Its configuration rule limits capacity to 100,000 with roll for its full policy.

The WestWing and EastWing examples that follow create histories in a Supervisor station.

WestWing403

Histories are exported using the following **`capacity`** and **`fullPolicy`** (defined in the configuration rule):

- AuditHistoryLog: unlimited and roll come from the **Default Rule**.
- LogHistory: 500 record sand roll: match no rule so the station uses the source history configuration, as in a remote controller station.
- Meter1: unlimited and roll come from the **MetersRule**.
- ZoneTemp1: 500 records and roll match no rule so the station uses the source history configuration, as in a remote controller station.
- Meter2: unlimited and roll come from the **MetersRule**.
- ZoneTemp2: 500 records and roll match no rule so the station uses the source history configuration, as in a remote controller station.

EastWing403

Histories are exported using the following **capacity** and **fullPolicy** (from configuration rule):

- AuditHistoryLog: unlimited and roll come from the **Default Rule**.
- LogHistory: 100,000 records and roll match the **EastWing** rule.
- Meter4: unlimited and roll come from the **MetersRule**.
- ZoneTemp1: 100,000 records and roll: match the **EastWing** rule.
- Meter3: unlimited and roll come from the **Meters Rule**.
- ZoneTemp2: 100,000 records and roll: match from the **EastWing** rule.

Again, the export function processes the rules in priority order. In this example, if we reorder the two optional rules by moving **EastWing** above **MetersRule**, our results would be different. Meter histories from EastWing403 would have a 100,000 record capacity.

BACnet and Rdbms

Import and Export descriptors configure importing (pull) from a station (usually a remote controller station) and exporting (push) data to another station (usually a Supervisor station).

An imported history effectively pulls data from a remote source station. For example, if a history import descriptor is under the **Histories** container of a **NiagaraNetwork→NiagaraStation**, the source is another history in the referenced remote station.

If a history import descriptor is under the **Trend Logs** of a **BacnetDevice**, the source is a BACnet Trend Log object residing in the BACnet device.

An exported history applies to a **NiagaraStation** (or database device under an **RdbmsNetwork**). An exported history exists in the local station, and is configured by a history export descriptor to push its data to a remote **NiagaraStation** (or RDBMS database). This adds it to the histories in that remote station (or to an RDBMS database).

Under a **BacnetNetwork**, the single **LocalDevice** component (that represents station data exported to BACnet) has a special view on its child **Export Table** component that permits exporting station histories as BACnet Trend Log objects.

You create history import descriptors and export descriptors under the **Histories** extension of a device using separate views.

External file history import

You may import delimited data from outside of a station and use the data to update histories. This feature (**FileNetwork**) is useful if the data originated in the system, were exported, manipulated outside the system and need to be imported back into a station database.

FileNetwork usage applies more to a Supervisor (PC) station than it does to a controller host, as it is more likely to contain text-delimited files needed for import as histories. In particular, it may be used with a Supervisor serving Energy Services reports to remote clients.

Specific requirements must be met by each delimited text file for successful import, namely:

- It must have a single timestamp column that contains both date and time. This means that if a delimited file has two columns: one for date and another for time, you must perform external upstream processing (outside of the system) before importing to combine these data into a single column.
- It must have a single column that contains the value required in the imported history. When configuring the history file import descriptor, you specify the **Value Format** as one of several types (`Numeric`, `String`, `Boolean`). **Value Facets** are also available in the descriptor.
- Optionally, it may also have a single column to use as **Status** in the history—however, this column must contain integer data (only) with values enumerated based upon status bits values.

The status import feature may be used mainly with data that first originated from a history that was exported to CSV, then subsequently manipulated outside of the station.

Adding file history import descriptors

To save engineering time, this procedure creates a single file import descriptor, which you can duplicate to create additional descriptors.

Prerequisites: You are working in Workbench connected to a Supervisor station. You have one or more Excel or other delimited files to import. You added a FileNetwork component to your station.

Step 1 Expand **Config→Drivers→FileNetwork→LocalFileDevice** and double-click **Histories**.

Step 2 To create a file import descriptor, click **New**.

The **New** window with the **`Type to Add`** property opens and click **OK**.

Step 3 Select the `ExcelCsvFileImport` type over the `DelimitedFileImport` type whenever the source file is a CSV type that was created using Microsoft Excel.

This provides more support for importing complex CSV-formatted data for example, including data that incorporates comma usage(s) within data fields.

The second **New** window opens.

Step 4 Configure the properties for this descriptor referencing a File that is patterned like the other delimited files you wish to import and click **OK**.

Step 5 To import the file, right-click the descriptor and click **Actions→Execute**.

Step 6 Adjust configuration properties of that descriptor until it executes (imports a history) properly without a fault status.

NOTE: The **`Timestamp Format`** property may take some tweaking.

Step 7 Duplicate your working import descriptor.

Step 8 Before executing the duplicate, change its **`File`** reference, **`History Id`**, and whatever other properties you wish to keep unique.

File import between stations

The following section describes how to import files from one station to another.

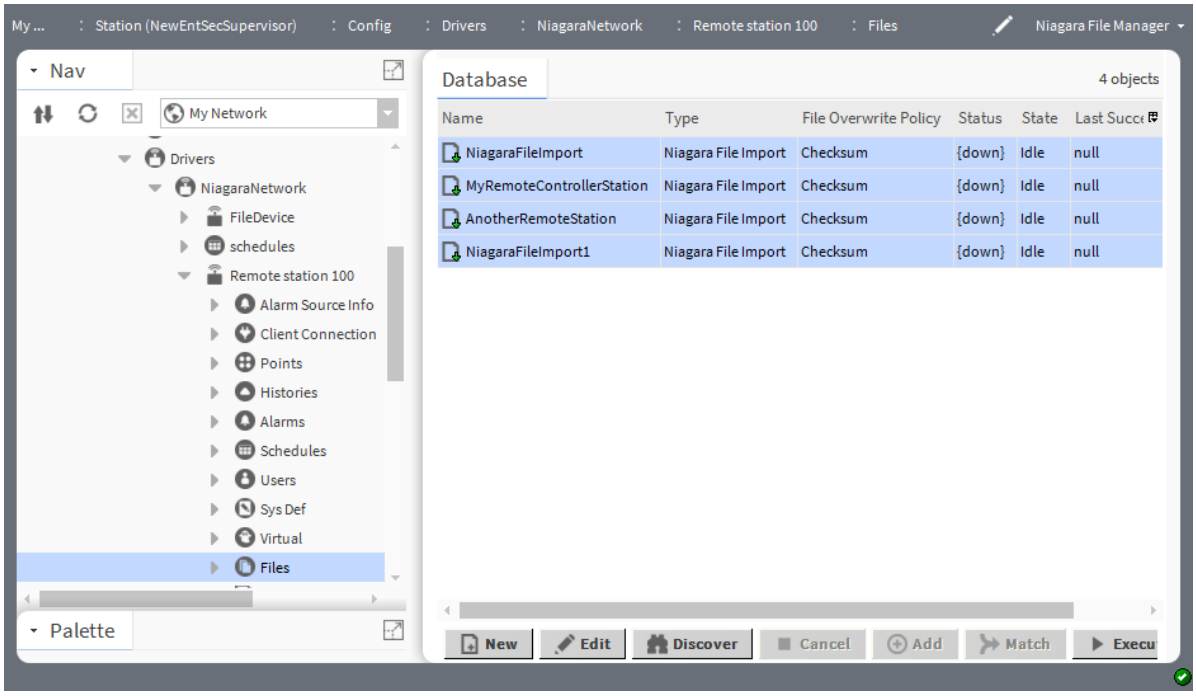
Discovering remote station files to import

You may import files from one station to another using a discovery job or set up a regular import job.

Prerequisites: You are connected to the station that will receive the imported file. The station has a Files node (NiagaraFilesDeviceExtension) already set up.

Step 1 Double-click the **Files** component under the **Files** directory list.

The **Niagara File Manager** view opens.



Step 2 To find the files to import, click **Discover**.

The discovery job finds the files on the other station and displays them in the **Discovered** pane. The **Database** pane shows files with existing file import descriptors.

Step 3 Select the files to import (an entire directory or a single file) and click **Add**.

You may use the **Ctrl** key to select individual files.

The **Add** window opens.

Step 4 Review the configuration and click **OK**.

Each file has a separate **`Files`** property to review and edit for the file import descriptor.

The station imports the files.

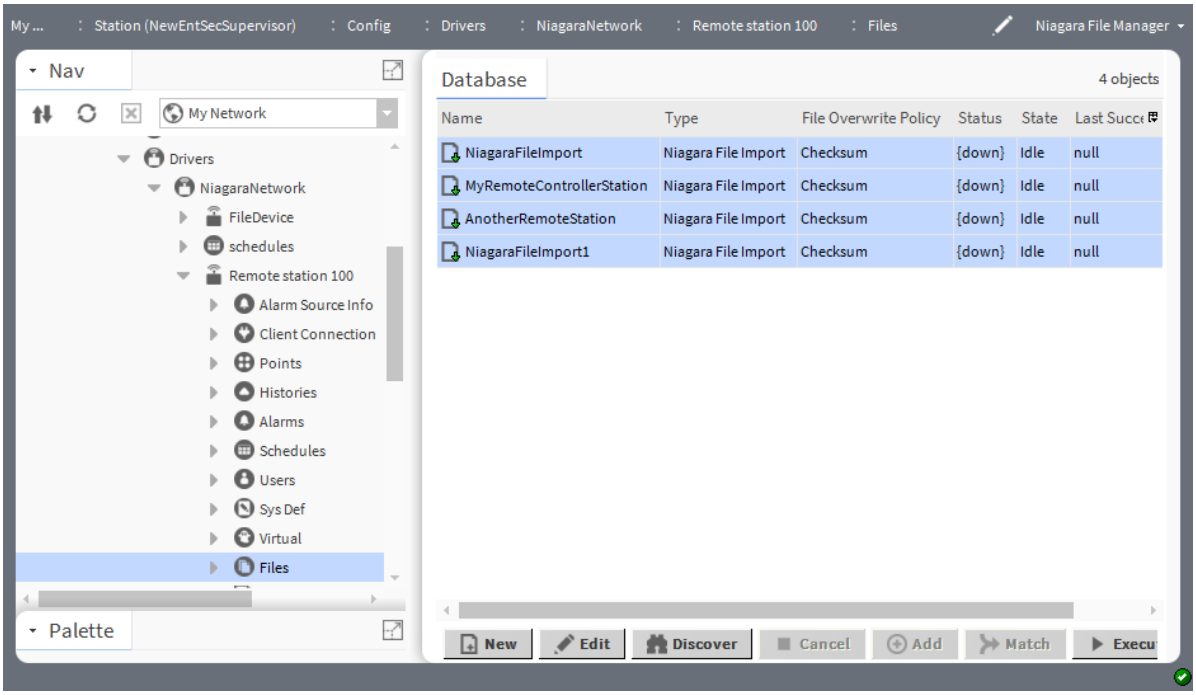
Setting up an automatic remote file import

On a regular basis you can configure an automatic import of files from one station to another.

Prerequisites: You are connected to the station that will receive the imported file(s).

Step 1 Expand **Config→Drivers→NiagaraNetwork**, expand a remote station and double-click the **Files** folder.

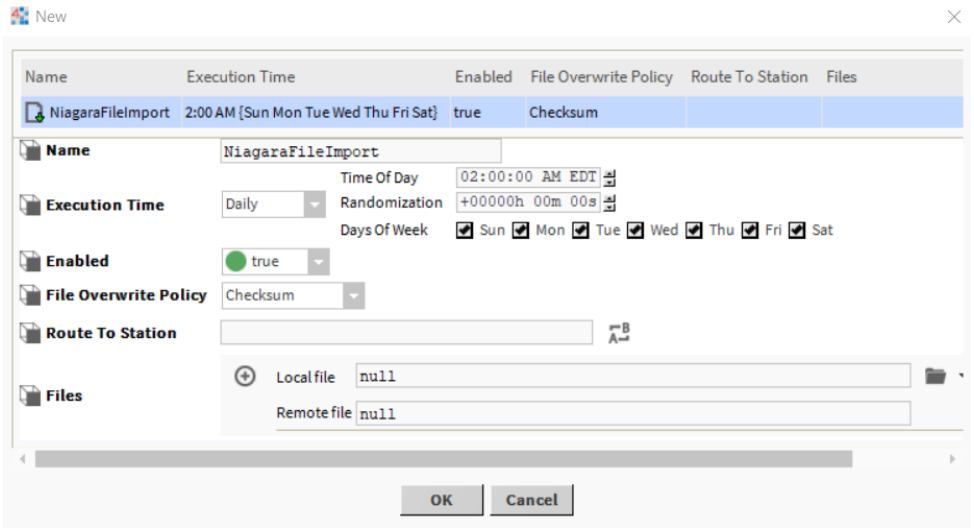
The **Niagara File Manager** view opens.



This manager lists the existing automatic file import jobs.

Step 2 To set up an automatic import, click **New**, select `Niagara File Import` from the drop-down list and click **OK**.

The **New** window with configuration properties opens.



By default the import instance is enabled. **`Files`** defines a pair of local and remote files. The **`Local file`** receives the import sent by the **`Remote file`**.

As of Niagara 4.13 **`Route To Station`** optionally specifies a route of intermediate station names (delimited by semi-colons) to reach the remote, reachable NiagaraStation from which to perform the file import operation.

Step 3 Configure at least when to perform the import (**`Execution Time`**).

Step 4 To set up a local file, click the down arrow beside the file folder ( ) to the right of the **`Local file`** property, select `File Ord Chooser`, locate the file and click **Open**.

Step 5 To define the source (remote) file for the import, enter file path in the **`Remote file`** property.

Step 6 To set up another local and remote pair of files, click the plus icon ( ) and identify the files.

Step 7 To delete a local and remote file pair, the X icon ( ).

The station removes the pair.

Step 8 After defining one or more file pairs, click **OK**.

The station saves the file descriptor.

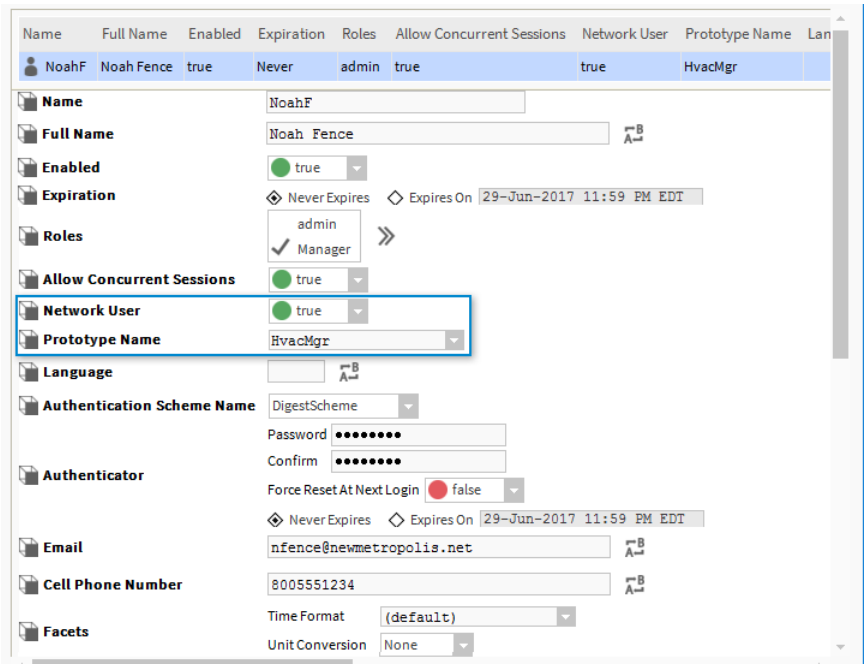
If you selected directory, the execution imports the entire contents of the directory including all contained files plus subdirectories and files. If the subdirectories and files already existing, the **`File Overwrite Policy`** must be met before each write.

Network users

In a large system, managing users can be time consuming, especially if you must make the same change to user records in each (separate) station. A class of users called network users permit centralized management of users in a multi-station system.

Two properties in a user component under the **UserService** identify a user as a network user.

Figure 24 User properties related to network user



- **`Network User`** is a Boolean toggle for identifying the user as a network user.
- **`Prototype Name`** identifies the prototype used by the network's synchronization strategy to implement user changes across a network.

NOTE: Network users, which the standard **UserService** in each station supports, are different from LDAP users, which a centralized LDAP or Active Directory server manages. For information about LDAP users, refer to the *Niagara LDAP Guide*.

Network users and the NiagaraNetwork

You use the **UserService** to add, modify, and delete users and the **NiagaraNetwork** to automatically replicate (or synchronize) the updated user data with other station(s).

At the same level as the **Points**, **Histories**, **Schedules**, and **Alarms** folders under a **NiagaraNetwork**'s **NiagaraStation** is a user device extension (**Users**). This extension enables and configures network user synchronization in and out of the station in relationship to the station's **NiagaraNetwork**.

No special view, apart from its **Property Sheet**, manages each user device extension nor is it a container for other components.

The **User Sync Manager**, accessed by right-clicking the **NiagaraNetwork** and clicking **Views→User Sync Manager**), provides an aggregate look at all user device extensions. Each row represents a **NiagaraStation**. Columns display each station's user synchronization properties.

Under each user device extension is a **Sync Strategy** property, which defaults to `Prototype Required`. The alternative to `Prototype Required` is `Use Default Prototype`. This option is less desirable for user synchronization.

User synchronization

Synchronization using network users solves the problem of how to change user properties in a large system with multiple, separate stations, each of which has the same set of users. Network users under the **NiagaraNetwork** facilitate centralized user management.

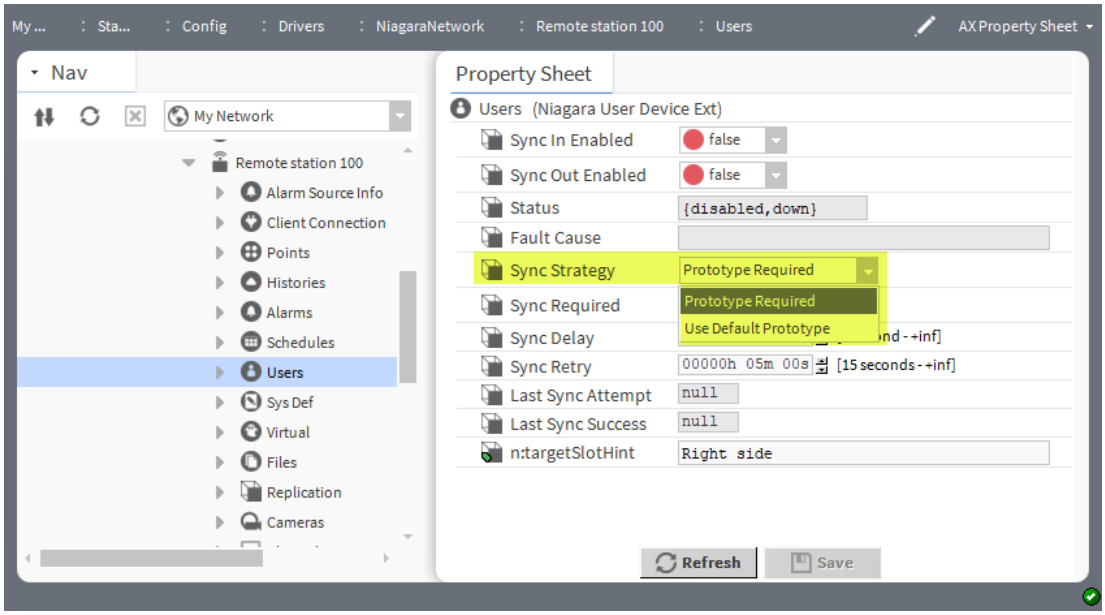
User synchronization requires these prerequisites and is limited by these factors.

- Both Supervisor and controller stations must use compatible password storage mechanisms. Otherwise, user synchronization fails and the corresponding **NiagaraStation**'s users device extension reports `fault`.
- Niagara 4 supports synchronize–in and synchronize-out between N4 stations (N4-to-N4). However, when synchronizing between N4 and AX stations, you can only synchronize-out from an N4 station to an AX station (N4-to-AX).
- Roles are not created during the synchronization process. For any role assigned to a network user in the sending station, you must set up a matching role on the receiving station.
- If a **NiagaraStation** reports a `one-way password hashing support mismatch`, the remote controller station is running a version of Niagara that is earlier than AX-3.7U1. Migrate the remote controller station to N4 before you configure and attempt user synchronization.

Users synchronization strategy

Two strategies are available for synchronizing users. The **`Sync Strategy`** property on the user device extension in the receiving station selects the strategy to use.

Figure 25 Sync Strategy property on the User Device Ext Property Sheet



A strategy of `Prototype Required` replicates (synchronizes) user data from the sending station to the receiving station when each **UserService** has an identically-named user prototype.

The `Use Default Prototype` strategy replicates user data from the sending station to the receiving station using the default prototype. You would use this strategy if a receiving station does not have a matching-named user prototype.

What happens in the sending station

The sending station is usually a Supervisor station.

In a sending station, user prototypes are important only in name. This is where you duplicate the default prototype and rename each duplicate uniquely. No users use these property values—whether a user is local only to the Supervisor or specified as a network user with a **`Prototype Name`**.

This is what happens on the sending side:

1. All user device extensions with **`Sync Out Enabled`** equal to `true` receive notification that a user event has occurred. This sets **`Sync Required`** equal to `true` in each user device extension.
2. The **`Sync Delay`** property determines how long to wait between a user event and a synchronization event. Each event resets the delay. That allows multiple changes to be made without necessarily requiring multiple synchronization processes.
3. After the sync delay passes, the Supervisor initiates the synchronization process with the subordinate station.
4. When complete, it sets **`Sync Required`** back to `false` and updates the synchronization timestamps.

What happens in the receiving station

In a receiving station, a user prototype must match the prototype name in the sending station.

The user prototype provides the **`Sync Strategy`** option and these local-override properties:

- **`Permissions`** (either a permissions matrix of local categories and rights or a Super User)
- **`Nav File`** (referencing a specific nav file under the local station's file structure)
- **`Web Profile`** (web-related properties)

When a network user is added or modified in a sending station, the receiving station uses its configuration of these two properties instead of those same properties in the source network user. You can configure prototypes for other local overrides.

This is what happens on the receiving side:

1. The subordinate station checks to see if **Sync In Enabled** of its proxy for the Supervisor station is set to `true`. If it is `false`, the subordinate station aborts the synchronization process.
2. The subordinate coordinates with the Supervisor to determine which users require synchronization.
3. The subordinate receives a list of out-of-sync user records from the Supervisor.
4. For each user, the subordinate station:
   - Finds a matching prototype among the **User Prototype**s. A prototype is either matched by name or, if **Sync Strategy** is set to `Use Default Prototype`, it uses the default prototype. If it finds no matching prototype, it ignores the user.
   - Overwrites the local override properties for the incoming user. These properties from the prototype include: **Permissions**, **Nav File**, and **Web Profile**.
   - Adds the new user or updates an existing user.
5. When complete, the station returns **Sync Required** to `false` and updates the synchronization timestamps.

Synchronizing user data

Replicating users and user configuration properties in all network stations is more efficient than manually setting up users in each station. This procedure explains how to synchronize user properties in the Supervisor station with users in a remote station.

Prerequisites: You set up user prototypes and users in the UserService. You are working in Workbench and are connected to a local Supervisor station.

Step 1 Expand **Config→Drivers**, double-click the **NiagaraNetwork** and click **Discover**.

A discovery job identifies all subordinate stations in the network.

Step 2 Select all discovered stations and add them to the Supervisor station's database by clicking **Add**.

The station automatically creates a set of **NiagaraStation**s under the **NiagaraNetwork**, one for each discovered subordinate station.

Step 3 Expand each **NiagaraStation→Users** node in the Nav tree and set at least **Sync Out Enabled** to true.

Synchronizing-out replicates user data from the Supervisor station to each target station. You would enable both **Sync In Enabled** and **Sync Out Enabled** in a subordinate Supervisor station.

Each user device extension defaults to a **Sync Strategy** of `Prototype Required`.

Step 4 To use the default prototype for synchronization instead of a user prototype, open each **Users** node under each **NiagaraStation** and change **Sync Strategy** to `Use Default Prototype`.

Step 5 Right–click the **NiagaraNetwork** in your Supervisor station and click **Views→User Sync Manager**.

The **User Sync Manager** view opens.

Step 6 Select the **NiagaraStation**s to synchronize and click **Sync**.

The system accesses the values for user properties from the user prototype names in the Supervisor station and replicates them to the remote station users that have the same user prototype names. If a user does not exist in a target remote station, the synchronization creates it.

Step 7 Expand each **NiagaraStation** under the **NiagaraNetwork** in the Supervisor station and double-click **Users**.

Step 8 Confirm that synchronization succeeded by reviewing **`Last Sync Attempt`** and **`Last Sync Success`** properties.

Specifying additional local-override properties

Three local properties in the station's default or user prototype override the same three properties received for any network user. These local-override properties are: **`Permissions`**, **`Nav File`** and **`Web Profile`**. Follow this procedure to configure other user properties as local-override properties.

Prerequisites: You are working in Workbench and are connected to a remote controller (receiving) station.

Step 1 To display the prototype properties, expand **Config→Services→UserService→User Prototypes** and double-click the **Default Prototype** or the user prototype.

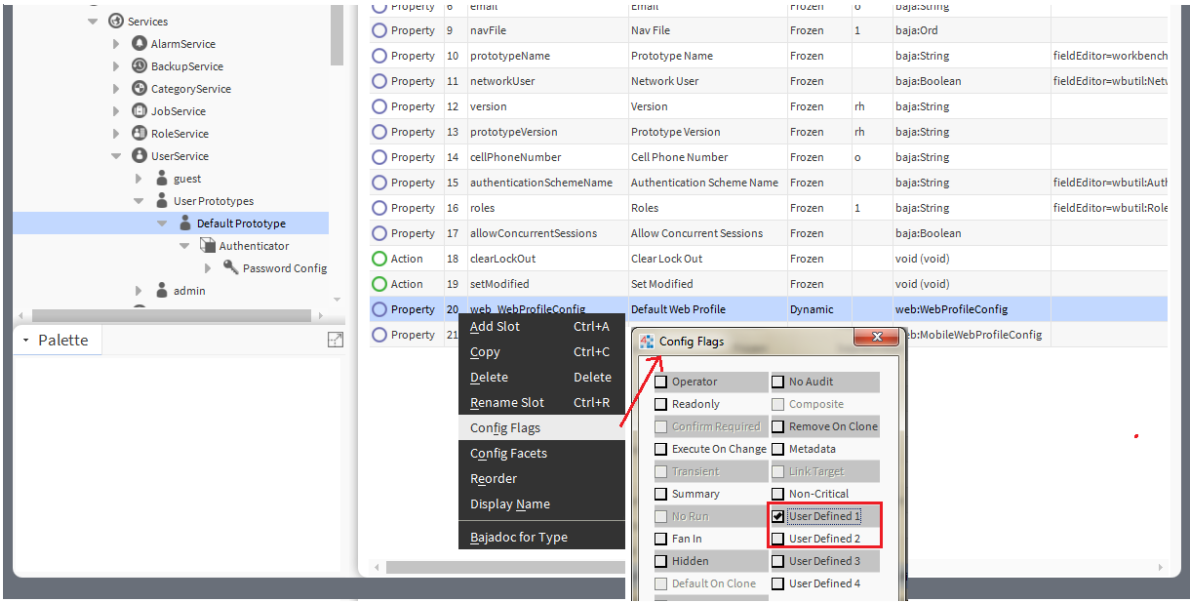
The **AX Property Sheet** for the prototype opens.

Step 2 To access to the component's slot sheet, use the menu in the upper corner of the sheet or right-click the prototype and click **Views→AX Slot Sheet**.

The **AX Slot Sheet** opens.

Step 3 Locate a property to include in the local-override properties, right-click it and click **Config Flags**.

The **Config Flags** window opens.



Step 4 Click to enable the **`User Defined 1`** flag and click **OK**.

Step 5 Notice that in the slot sheet view, the flags column now includes a 1 for the property. This is similar to the **`permissions`** and **`navFile`** slots.

When a network user synchronization occurs for a user that references this prototype, all properties with this flag set use the local values as overrides.

Sync-in and –out user device extension configurations

Typically, a Supervisor station sends user data and multiple remote controller stations receive the synchronized data. Data in and out is flexible. This topic provides examples of how to configure the **`Sync In Enabled`** and **`Sync Out Enabled`** properties of the user device extension of **NiagaraStation**s.

Supervisor and subordinate configuration

Figure 26 Supervisor station configuration

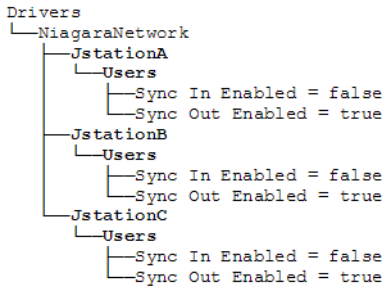
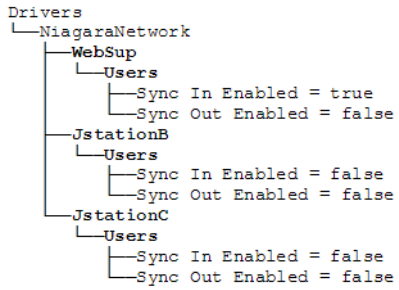


Figure 27 Remote controller station configuration



Multi-tier configuration

It is possible to have a multi-tier network user synchronization configuration with a single master Supervisor host and several subordinate Supervisor hosts, each with multiple subordinate remote controller stations. In the **NiagaraNetwork** of each subordinate Supervisor station, **`Sync In Enabled`** would be set to `true`. This sends user data from the master station to the single **NiagaraStation** in the subordinate Supervisor that proxies the master Supervisor.

In the subordinate Supervisors' other **NiagaraStation**s (the ones that proxy remote controller stations), **`Sync Out Enabled`** is set to `true`. In this configuration, each subordinate Supervisor station is both a receiver and a sender of user data.

Figure 28 Master Supervisor station configuration

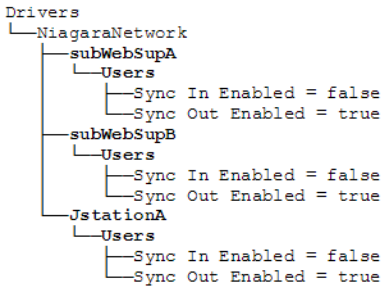


Figure 29 Subordinate Supervisor station A configuration

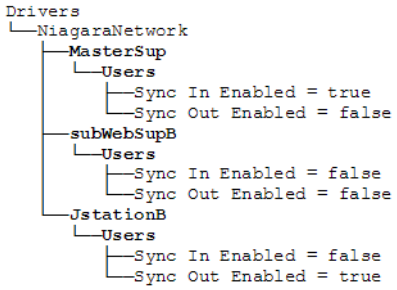
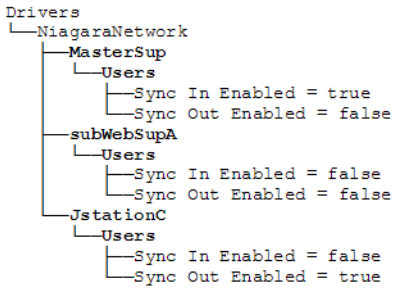


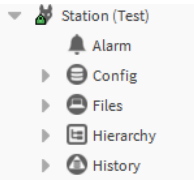
Figure 30 Subordinate Supervisor station B configuration



About virtual component spaces

A station's virtual space describes a transient location for modeling components, devices and points that are not saved to the station database. The original requirement for virtual objects, with a persisted gateway component to access them, came about from driver use-case needs.

Figure 31 Top level object spaces



Object types in these spaces are:

- Components — in the component space under a **Config** node (regular component space).
- Files — in the files space under a **Files** node.
- Histories— in the histories space under a **History** node.

You can optionally enable virtual components to provide a virtualized Px view of any Px views on the remote (actual) component. The Px files are imported (on demand) from the remote device to the Supervisor when the virtual Px view is first accessed.

A station uses these objects when running and, when the station stops, they persist as components in the station's database, files in the station's directory and histories in the history database. Virtual component spaces are different from regular components, in the following ways:

- You may have multiple virtual component spaces, each belonging to a different virtual gateway (specialized persistent components in the station's component space (under **Config**).
- A virtual component space is a mapping of virtual components organized in a tree fashion, created at runtime when components under its virtual gateway, and accessed when subscribed.

- Virtual components are transient components created in the station only when needed. When no longer necessary (that is when unsubscribed), the driver automatically removes from the running station. This permits monitoring applications in cases where proxy points would use too many resources.
- Virtual components have limitations and should not be confused with other components, such as proxy points. Links to and from virtual components, use of point extensions (history, alarm, etc.), and typical copy/paste operations are not supported.

Virtual component licensing

Virtual components under a **NiagaraNetwork** are a standard part of a Supervisor host's license.

This feature is licensed with the Boolean virtual attribute in the **`niagaraDriver`** feature line:

```
<feature name="niagaraDriver" expiration="never" device.limit="none" history.limit=
"none" point.limit="none" schedule.limit="none" virtual="true" parts="AX-DEMO"/>
```

This option is typically not licensed in any controller host. It does not need to be.

Virtual objects

Dynamically-created virtual objects provide monitor access to the entire component structure of each remote controller station without the overhead (or engineering necessity) of using proxy points. They provide write access to most component properties. A **NiagaraStation** under a Supervisor station contains the **Virtual** container with these object.

A small virtual ghost icon superimposed in the lower right over a normal component icon reminds you that you are looking at a virtual component.

The large scale application for virtual components is in a Supervisor station where proxy point values in remote controller stations can be modeled, graphically presented on Px pages hosted by the Supervisor and then deleted when they are no longer needed. A station in a controller may also have a few proxy points, typically in cases where remote data are required within its control logic. However, proxy points are largely synonymous with a Supervisor station.

Virtuals provide an attractive alternative to creating proxy points when engineering a Supervisor station—especially for general Px page presentation. This feature dramatically reduces the general need for proxy points.

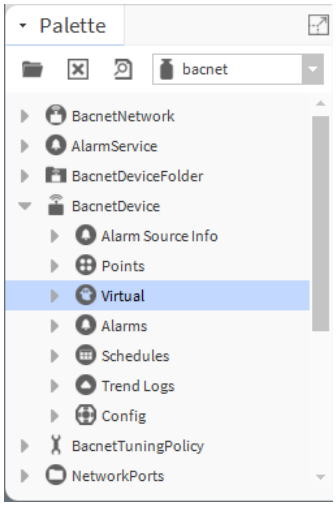
Virtual gateway

Every **NiagaraStation** component in a station's **NiagaraNetwork** has a frozen **Virtual** (virtual gateway) slot at the same level as that station's device extension slots (**Points**, **Schedules**, etc.). Virtual gateways persist in the station database along with other persisted components. Each virtual gateway provides the link between the normal component space and its own virtual components.

Virtual node under a NiagaraStation

In the initial driver application of virtual components, each device component in a driver network has its own single virtual gateway (as shown here) for a **BacnetDevice** in the **`bacnet`** palette.

Figure 32 One virtual gateway per device



This is a logical division of gateways (separating data by its source device). Future virtual component applications may not always use this one-to-one device hierarchy. At some future point, a driver may have a network level virtual gateway or possibly provide multiple virtual gateways under the same device level or network level.

A few drivers have device-level components that contain a **Virtual** gateway child. This is in addition to the standard collection of slots for device-level components. In the Niagara driver architecture, a device's virtual gateway provides access to virtual components in the station's virtual component space, specific to the device.

The BACnet driver, and the **`niagaraDriver`** (**NiagaraNetwork**) both provide virtual components.

Successfully accessing components under the **Virtual** gateway dynamically adds them as virtual components (or virtuals) while they are subscribed, but they exist only in memory. They do not persist in the station database like proxy points.

Virtual components are sometimes referred to as virtual points, but they are quite different from the proxy point components found in most drivers. The term "virtual point" is a actually a misnomer as virtual components represent any component type in the source remote station (not just control points).

NOTE: Baja virtual components are not specific to only drivers. Other core components use them.

Niagara virtual gateway requirements

Virtual gateways provide virtual component access only if the following are all true:

- The local station's host is licensed for a virtual gateway. Typically, licensing applies only to a Supervisor host. Virtual gateways do not apply to controller stations.
- The parent **NiagaraStation** component is enabled and its property **`Virtuals Enabled`** is set to `true`. By default, this property is set to `false` for station security reasons.
- The host running the remote station is running a recent version of Niagara

Virtual applications and advantages

Subscribing to a persistent component only to view a Px page followed by unsubscribing when no one is looking at the page consumes valuable station resources. Using virtual components, a station not only unsubscribes from the virtual component, but it also frees up station resources when it removes the component from the station's memory.

A driver that offers virtual objects provides two main applications: Px view bindings for the simple polling of values without the station resource overhead of persisted proxy points, and quick review and (if possible) adjustments to one or more properties for objects in native devices.

Low overhead Px bindings

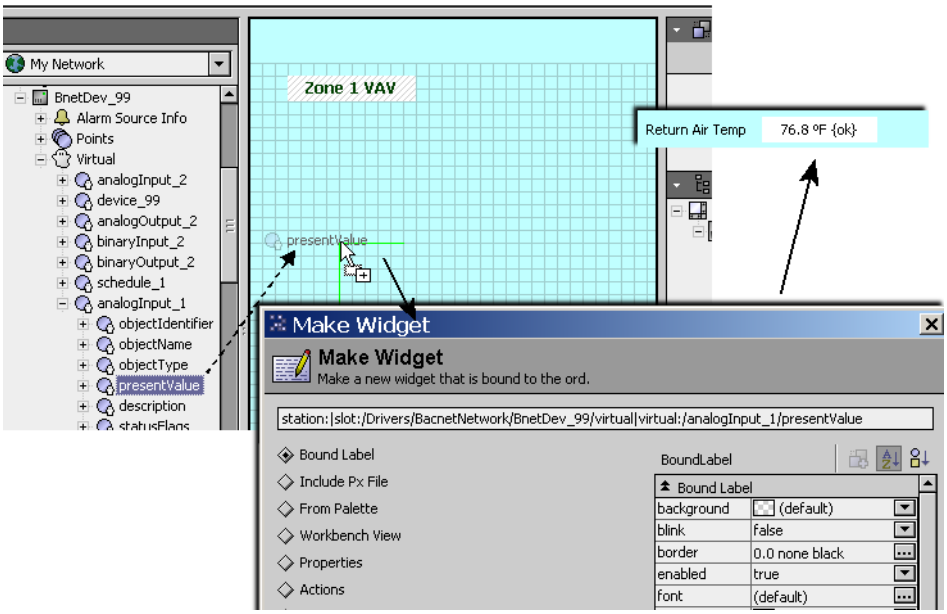
Virtuals in a Supervisor provide the same on-demand subscription of remote real-time data on Px pages as do proxy points, including right-click popup menus to issue actions. However, the Supervisor station overhead from scores of persisted proxy points is gone. Instead, the only persisted items are the ORDs to virtual objects within Px widgets. This results in fewer resources consumed, along with faster station startup.

Upon being unsubscribed, the poll scheduler removes virtual components from the scheduler (or subscription mechanism) and station memory. The only persisted (permanent) records of a virtual object are its ORDs (using virtual syntax) in the Px widget bindings. In some cases, particularly with replicated device applications, this allows a controller station to graphically monitor many more devices than it can monitor using only proxy points.

NOTE: A virtual gateway cannot have its own Px view, but you can use its child virtual components in Px views for other components, for example on the device component itself or its **Points** extension, and so on.

The Px bindings to virtual objects use virtual ord syntax that includes the virtual gateway within it (and activates it at runtime). Dragging a virtual component to the Px page automatically resolves the ORD and makes your selection in the popup Px **Make Widget** window, as shown here.

Figure 33 Dragging a BACnet virtual component into a Px editor view with the resulting Make Widget popup



Depending on the driver, Px usage details may differ for virtual components. For example, usage of virtual components in Px views includes special considerations.

Time savings

Virtuals provide a possible savings in engineering time as point discovery and creation using the Bql Query Builder is not needed. Instead, virtual components expand under a **NiagaraStation**'s virtual gateway to reflect the entire component tree for the station. From the tree you can drag virtuals to Px pages and select appropriate settings in the popup **Make Widget** windows.

Quick property change reviews

Using the Workbench property sheets to make property changes to virtual components can speed what-if scenarios. Otherwise, you would need to create proxy points, then delete them after reviewing and changing properties. This application applies to both **NiagaraNetwork** virtual components and BACnet virtual components.

Easy user access

Writable virtuals can easily provide user access to almost any property in a Px page graphic for both display and adjustment purposes. For example, to expose the high alarm limit for a point, a user can expand its alarm extension to reveal child containers, drag the **`Offnormal Algorithm`** virtual to the Px page, and in the **Make Widget** popup window, select the ORD to the **`High Limit`** property. Creating a proxy point for this purpose beforehand is not needed.

Virtual limitations and guidelines

Virtuals are transient. They do not persist. The station dynamically creates and subscribes to them only when you access them. They are not permanently stored in the station database. This precludes any linking to or from virtuals as links would be lost. Nor are point extensions (alarm, history) supported for virtual components. These features require the use of proxy points, which are persisted in the station database.

Limitations

Values from remote stations needed within station logic require proxy points. For example, a math **Average** object that averages zone temperatures originating in multiple controller stations requires a proxy point for input to define the relevant (remote station) data source.

Point extensions under virtuals are not supported. This includes alarm and history extensions, although from a best-practices perspective, such extensions are often misplaced in proxy points.

Using virtuals on Px pages is recommended only after finalizing most other component configuration, including finalzing facets and the config flags of any (target) control points that will be incorporated. The reason is that the virtual ORD syntax used in a Px widget binding to the point actually includes that point's facets and config flags (used at the time of widget creation), encoded into the (persisted) ORD itself. So, if a target point has a subsequent facets change, a Px page with a virtual binding to it may require re-engineering.

Guidelines

To summarize, here are some quick application guidelines for virtual components versus proxy points:

- To link station logic in to or out of the data item, use a proxy point.
- To alarm or trend/log (history) a data item, use a proxy point.
- To use the value of a data item only while a user is looking at a Px view, use a virtual component.
- To configure values in the device for one-time commissioning, use a virtual component.

Activating a virtual gateway

The virtual gateway under the **NiagaraNetwork** is in a station component (**Station** and **EdgeStation** components).

Prerequisites: You are working in Workbench connected to a Supervisor station.

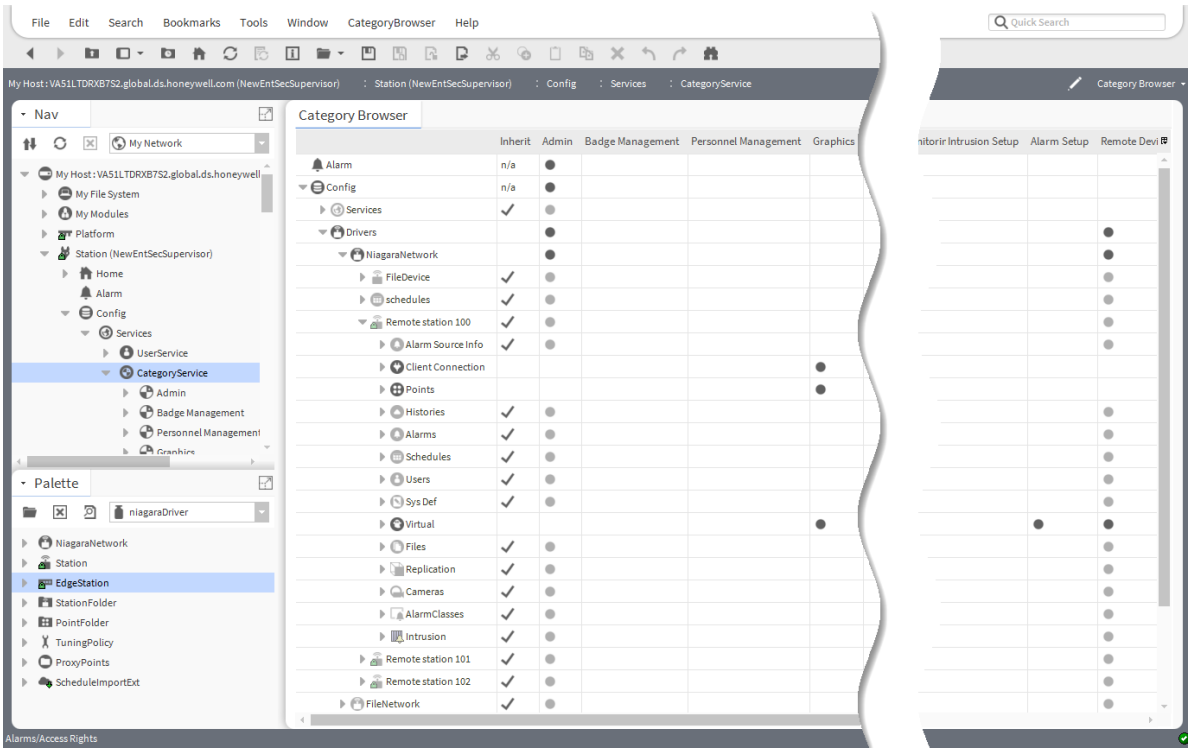
Once you enable a virtual gateway, any Supervisor station user with admin-level permissions can view all users in each remote station's **UserService** by expanding the **Property Sheet** of the virtual component for the **UserService**. The same user can expand the remote station's **PlatformServices** as a virtual component and access critical properties and child services including an action to restart station/reboot host!

Therefore, before enabling any Virtual gateway, you should restrict user access by assigning a restricted category to the virtual gateway. Then, on each virtual component under that gateway, assign the appropriate category(ies) before re-assigning less restrictive category(ies) to the parent gateway.

NOTE: The Supervisor station maintains the categories assigned to active virtual components even after removing them from the station's memory cache.

Step 1 Expand **Config→Services** and double-click **CategoryService**.

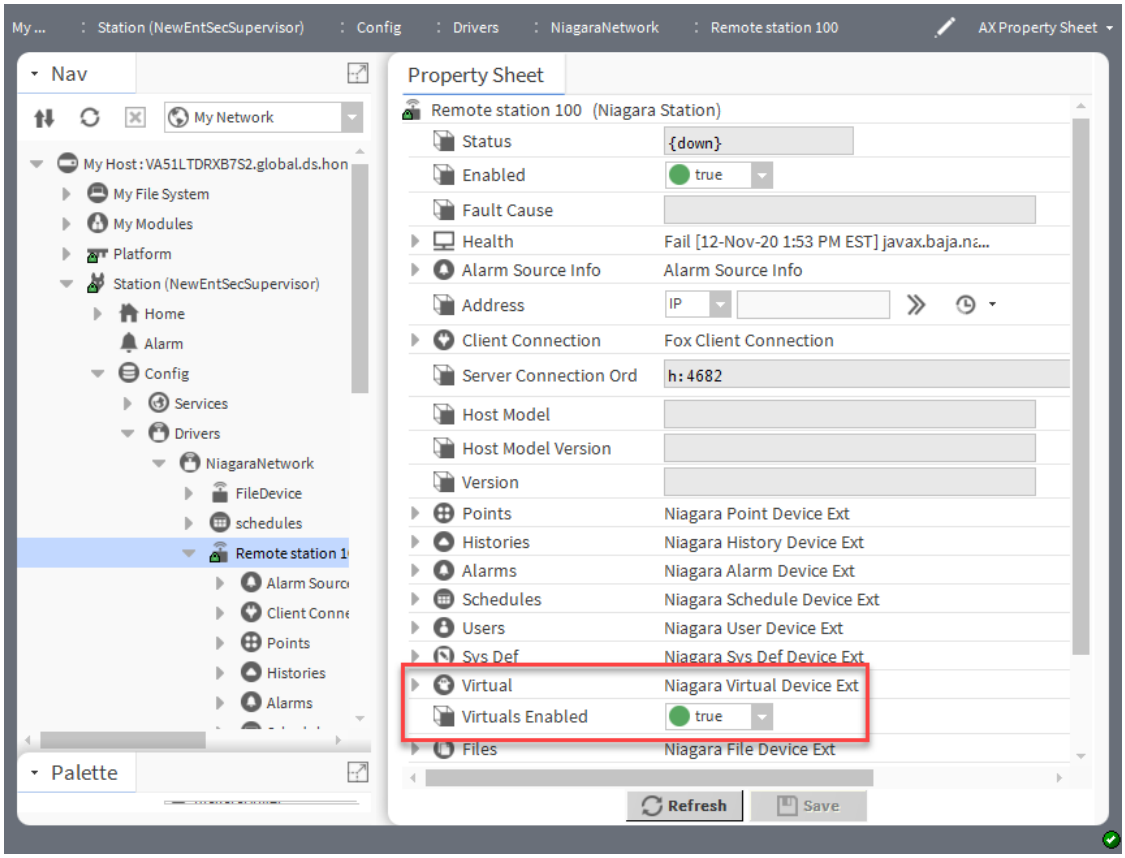
The **Category Browser** opens.



Step 2 Expand the **Config** row in the **Category Browser** to expose the virtual gateway, configure the allowable categories and click the save tool ( ).

Step 3 To enable the use of the virtual gateway, expand **Config→Drivers→NiagaraNetwork** in the Nav tree and double-click a station component.

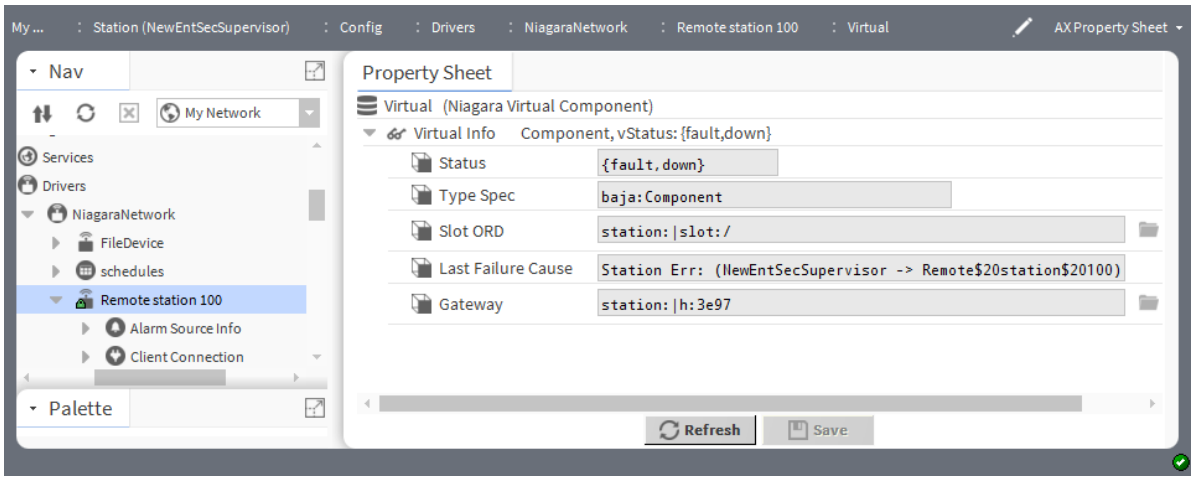
The station component's **AX Property Sheet** opens.



Step 4 Change **`Virtuals Enabled`** to `true` and click **Save**.

Step 5 To open the gateway, double-click the **Virtual** device extension or expand it in the Nav tree.

The driver gathers data from the device. The results appear as child virtual components.



Unlike most device extensions, there is no special Workbench view for a virtual gateway.

Step 6 If you expanded the **Virtual** node, you can further expand each object to view its virtual properties including its value and poll status.

Virtual ORD syntax

The general syntax for a virtual ord uses a specialized form.

The ORD syntax is:

`<ord to VirtualGateway>|virtual:/virtualPath` where:

`virtualPath` represents some hierarchy of virtual components and child properties.

For example, for BacnetVirtualComponents, the general virtual path syntax is:

`<ord to VirtualGateway>|virtual:/objectType_Instance/propertyName`

Or in the special case of an arrayed property:

`<ord to VirtualGateway>|virtual:/objectType_Instance/propertyName/elementN`

where `N` is the property array index.

For specific virtual ord details, refer to the corresponding driver document.

NOTE: In Niagara 4, virtual components in a **NiagaraNetwork** use a much simpler virtual ord syntax than in previous releases, due to a new virtual cache mechanism. Much information that was formerly in the `virtualPath` portion of the ord syntax now resides in this cache.

On Px pages, virtual components, which build on standard Baja virtual component syntax further append facets and config flags. This creates complex ords that the station automatically resolves when you drag a virtual onto a Px page. As a result, you may notice a very wide Make Widget window.

For example, a single ord may look like this:
`station:|slot:/Drivers/NiagaraNetwork/J202_TestG/virtual|virtual:/ Drivers;d;c;driver%3aDriverContainer/M2mIoNetwork;d;c;nrio%3aM2mIoNetwork/ LocalIo16;d;c;nrio%3aNrio16Module/points;f;c;nrio%3aNrio16Points/ RTU1;d;c;nrio%3aNrio16PointFolder/Inputs1;d;c;nrio%3aNrio16PointFolder/ ui1%242dReturnAir;d;c;control%3aNumericPoint|slot:out$3bf$3bp$3bbaja$253aSta tusNumeric$3b267$3bunits$253du $253afahrenheit$253b$25baF$253b$2528K$2529$253 b$252a0$252e5555555555555556 $252b255$252e3722222222223$253b$257cprecision$2 53di$253a1$257cmin$253dd$253a $252dinf$257cmax$253dd$253a$252binf$257cnVirtua lW$253db$253atrue/value`

Obviously, such ORDs are difficult to decipher and safely change in a text editor. The Baja text property editor within the Px Editor normalizes the ORD portion to appear like other component ORDs.

Setting up virtual Px view support

Virtual Px support is disabled by default. You can enable it on a Supervisor station.

Prerequisites: You are using Workbench and are connected to a Supervisor station.

Step 1 Expand **Config→Drivers**, right-click **NiagaraNetwork** and click **Views→AX Property Sheet**.

The **NiagaraNetwork Property Sheet** opens.

Step 2 Expand **Virtual Policies** and set **`Import Virtual Px Files On Demand`** to `true`.

Once enabled, any virtuals for which the remote (actual) component has a Px view provide a virtualized Px view on the Supervisor.

Viewing Virtual Px graphics on demand

A Px graphic on demand makes it possible for a Supervisor station to present graphics for viewing from a remote station on demand.

Prerequisites:

- You have an open connection to a Supervisor station via Workbench or a browser.
- You set up the system database on the Supervisor station.
- You enabled the **`Import Virtual Px Files On Demand`** property in the **`Virtual Policies`** on the **NiagaraNetwork**.

- You have existing components, such as devices, with Px graphics on the subordinate stations in your **NiagaraNetwork**.
- You have successfully indexed subordinate stations in your **NiagaraNetwork**, and virtuals must be enabled and set up properly for user permissions, etc.

Step 1 Open the **Search** sidebar and click the settings icon ( ).

The **Search Settings** window opens.

Step 2 Under the **Scope Selection** property, click **System Database** and click **OK**.

Step 3 In the **Search** sidebar, enter the NEQL query: `n:device` and click the search icon ( ).

The framework lists the devices it found.

Step 4 To open the virtual Px graphic on demand, in the search results, click on the name of a virtual component—one that has a Px graphic.

The virtual status displays as a Px graphic.

Opening a Scheduler view on a virtual component

Virtual Schedules in the **NiagaraNetwork** allow you to open a **Scheduler** view on a virtual component. You can make adjustments to the schedule as well.

Prerequisites:

- You have an open connection to a Supervisor station (via Workbench or a browser).
- You have existing schedules on the subordinate stations in your **NiagaraNetwork**.

Step 1 Open the **Search** sidebar and click the settings icon ( ).

The **Search Settings** window opens.

Step 2 Under the **Scope Selection** property, click **System Database** and click **OK**.

Step 3 In the **Search** sidebar, enter the NEQL query: `n:schedule`.

The search results display a list of all schedules that were pulled into the SystemDb. Note that the schedules are modeled as virtual components.

Step 4 To open the **Scheduler** view on the virtual component, click one of the schedules listed in the search results.

Step 5 Make adjustments to the schedule as needed.

Virtuals troubleshooting

This topic addresses typical issues when working with virtual components.

The virtual gateway in my Supervisor NiagaraNetwork reports {fault}.

The virtual gateway in a Supervisor station may be disabled or even in fault. Confirm that virtuals are licensed for your Supervisor station, confirm that **Virtuals Enabled** on the **NiagaraNetwork Property Sheet** is enabled (`true`), and check the **Last Failure Cause** on the virtual gateway (**Virtual** node under the **NiagaraStation**.

I want to configure a controller station to support virtuals. The Last Failure Cause under my NiagaraStation reports Niagara Virtual Components are not licensed.

A virtual gateway may report a fault status because, even though the station's **Virtuals Enabled** property is set to `true`, the host controller running this station may not be licensed for virtuals.

The values displayed for the components under a NiagaraStation do not look right.

If you are engineering a job while still configuring a remote station that is modeled by a **NiagaraStation** in the local **NiagaraNetwork**, you may need to right-click on the **Virtuals** node and click **Actions→Refresh** to update the data from the remote station.

Setting up the Station Manager to display index status

Global system indexing can be used to collect entity information from the **NiagaraStations** under the Supervisor station's **NiagaraNetwork**, which it saves in the Supervisor's system database. This makes it possible to query an entire system. This procedure sets up the columns of the **Station Manager** to report on the progress of subordinate station indexing.

Prerequisites: You have set up the SystemDbService and SystemIndexService under the Services folder in the Nav tree. Virtuals are enabled and configured for user permissions. You have successfully started to execute the global system indexer against subordinate stations under your NiagaraNetwork.

To learn more about setting up indexing itself, refer to the *Niagara System Database and System Indexing Guide*.

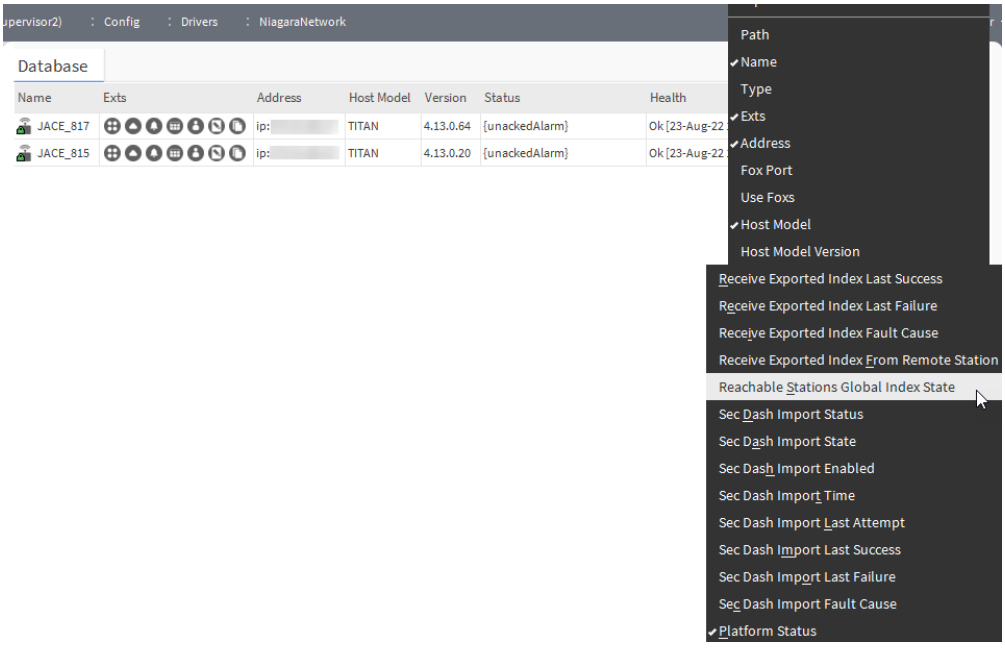
Step 1 To display system index progress information for subordinate stations, expand **Config→Drivers** and double-click the **NiagaraNetwork**.

The **Station Manager** view opens.

Step 2 Click the icon that expands the columns list ( ).

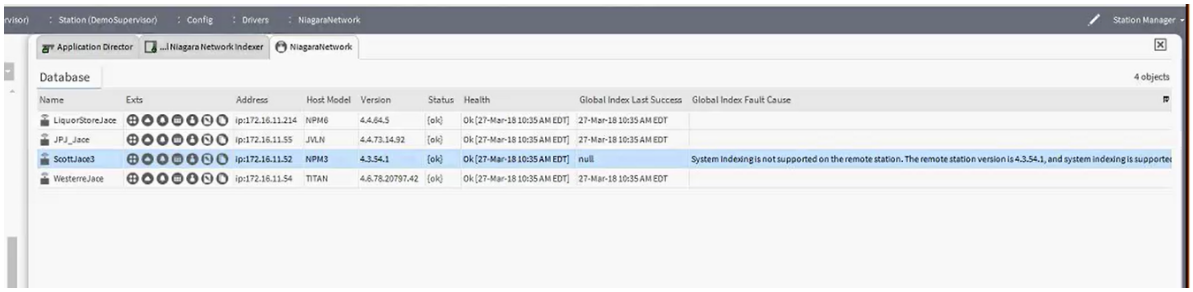
As of Niagara 4.13, to view the indexing progress of the downstream reachable stations, you can select **Reachable Stations Global Index State** from the list of optional columns.

This exposes a list of additional columns you can display.



Step 3 In the drop-down list, select the system indexing progress columns you wish to display in the **Station Manager**.

The additional columns appear in the view.

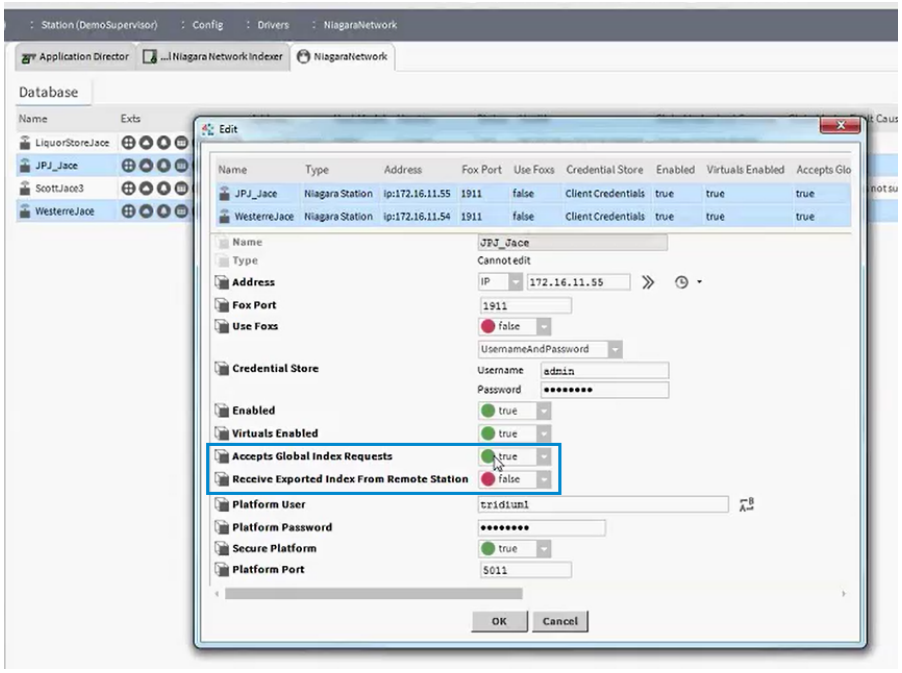


Step 4 To sort the displayed records by indexed column, click the column header.

For example, you could list the collected records by when System Indexing last occurred and when it last occurred successfully or unsuccessfully in each station.

Step 5 To batch edit (gang edit) more than one station at a time, select the stations to edit and click **Edit**.

The edit window opens.



The screen capture shows simultaneously editing two properties for two selected stations:

- **`Accepts Global Index Requests`** is set to `true` to include the stations in the global system indexing operation.
- **`Receive Exported Index From Remote Station`** is set to `false` to not allow unwanted export (push) to be accepted also for security reasons.

NOTE: The System Database runs only on Supervisor platforms, not on controller platforms.

Chapter 4 Components

Topics covered in this chapter

- driver-ArchiveFolder
- driver-ConfigRule
- driver-DelimitedFileImport
- driver-DriverContainer
- driver-ExcelCsvFileImport
- driver-FileDevice
- driver-FileHistoryDeviceExt
- driver-FileHistoryWorker
- driver-FileNetwork
- driver-HistoryNetworkExt
- driver-HistoryPollScheduler
- driver-PingMonitor
- driver-SendTimer
- driver-SendTimes
- driver-TuningPolicy
- driver-TuningPolicyMap
- fox-FoxClientConnection
- fox-FoxFileSpace
- fox-FoxServerConnections
- Server Connections (fox-ServerConnections)
- fox-FoxService
- fox-FoxSession
- niagaraDriver-BogProvider
- niagaraDriver-CyclicThreadPoolWorker
- niagaraDriver-LocalSysDefStation
- niagaraDriver-NiagaraAlarmDeviceExt
- niagaraDriver-NiagaraEdgeStation
- niagaraDriver-NiagaraFileDeviceExt
- niagaraDriver-NiagaraFileImport
- niagaraDriver-NiagaraHistoryDeviceExt
- niagaraDriver-NiagaraHistoryExport
- niagaraDriver-NiagaraHistoryImport
- niagaraDriver-NiagaraNetwork
- niagaraDriver-NiagaraPointDeviceExt
- niagaraDriver-NiagaraPointFolder
- niagaraDriver-NiagaraProxyExt
- niagaraDriver-NiagaraScheduleDeviceExt
- niagaraDriver-NiagaraScheduleImportExt
- niagaraDriver-NiagaraStation
- niagaraDriver-NiagaraStationFolder
- niagaraDriver-NiagaraSysDefDeviceExt
- niagaraDriver-NiagaraSystemHistoryExport
- niagaraDriver-NiagaraSystemHistoryImport
- niagaraDriver-NiagaraTuningPolicy
- niagaraDriver-NiagaraUserDeviceExt
- niagaraDriver-NiagaraVirtualDeviceExt
- niagaraDriver-NiagaraVirtualNetworkExt
- niagaraDriver-ProviderStation
- niagaraDriver-ReachableStationInfo
- niagaraDriver-RoleManager
- niagaraDriver-SyncTask
- ndriver-NDeviceUxManager
- ndriver-NPollScheduler
- serial-SerialHelper
- tunnel-TunnelService
- tunnel-SerialTunnel
- tunnel-TunnelConnection
- niagaraVirtual-DefaultNiagaraVirtualCache
- niagaraVirtual-NiagaraVirtualBooleanPoint

Descriptions included in the following topics appear as context-sensitive help topics when accessed by:

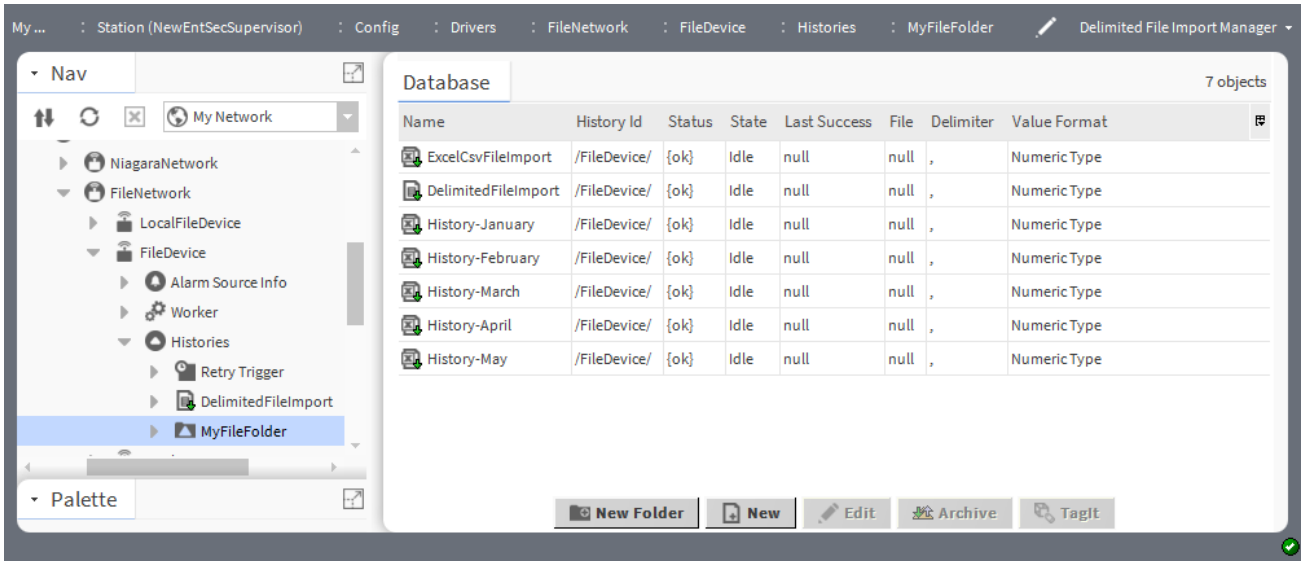
- Right-clicking on the object and selecting **Views→Guide Help**
- Clicking **Help→Guide On Target**

driver-ArchiveFolder

ArchiveFolder is a folder available to hold and organize history import descriptors and/or export descriptors.

Each archive folder has its own manager view Niagara History Import or Export Manager.

Figure 34 Archive Folder



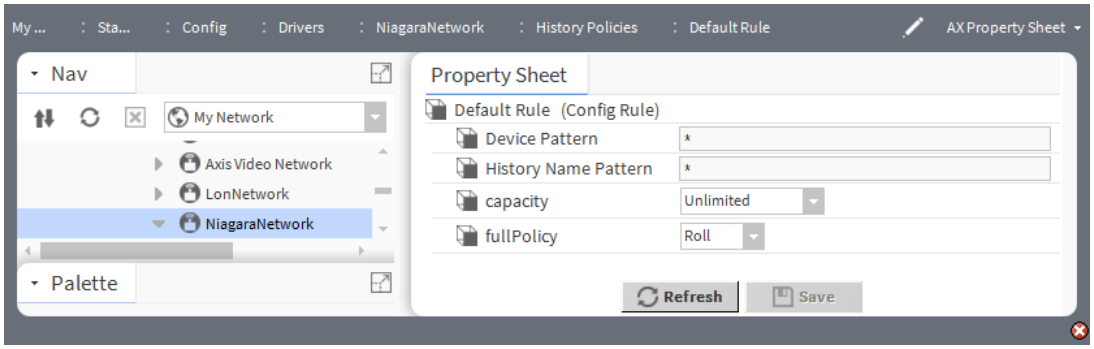
To select this view right-click **Histories**and click **ViewsNiagara History Import Manager** or **Niagara History Export Manager**.

To add one of these folders, expand **Config→Drivers→NiagaraNetwork→NiagaraStation** or **FileDevice**, double-click **Histories** and click **New Folder**.

driver-ConfigRule

This component of the **NiagaraNetwork** selects the histories to export from one or more remote stations and determines what happens in the receiving station.

Figure 35 Config Rule properties



To view these properties in a receiving station, expand **Config→Drivers** right–cllick the **NiagaraNetwork**, click **Views→AX Property Sheet**, expand **History Policies** and click **Default Rule**.

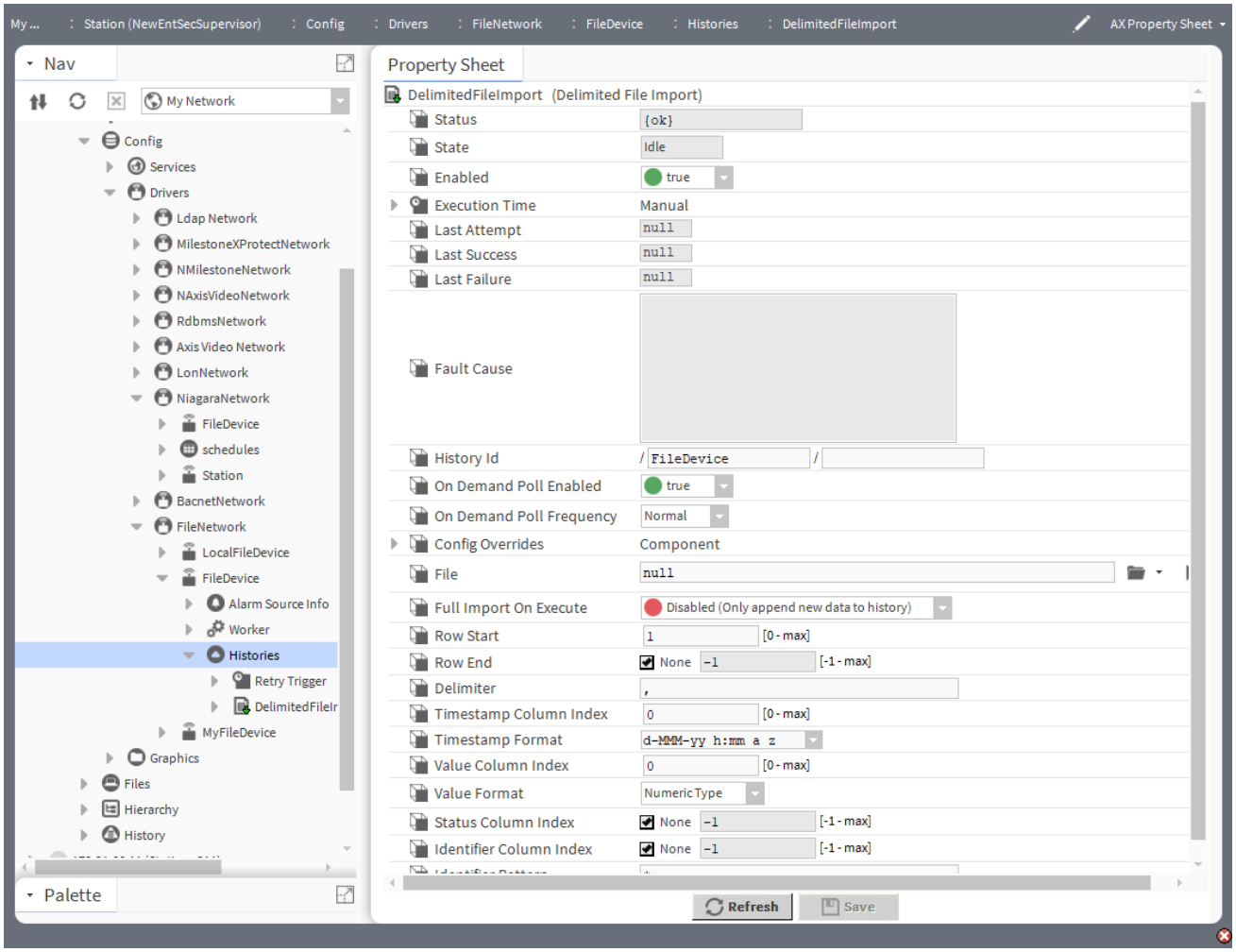
Property	Value	Description
Device Pattern	text (defaults to wildcard (*) which exports from all stations)	Identifies the names of station(s) with histories to export.
History Pattern Name	text (defaults to wildcard (*) which exports all histories)	Identifies the names of histories to export. **NOTE:** Both **`Device Pattern`** and **`History Name Pattern`** must apply for the rule to be used—otherwise, the receiving station evaluates the history in question against the next rule down (in order) under **`History Policies`**.
capacity	drop-down list	Specifies the number of trend log records (histories) to store in the histories database. When capacity is reached, newer records overwrite the oldest records.
fullPolicy	drop-down list	Specifies what happens when a trend log (history) reaches capacity. Applies only if **`Capacity`** is set to `Record Count`. When capacity reaches record count, the newest records overwrite the oldest records. `Roll` ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches capacity. Full policy has no effect if `Capacity` is `Unlimited`.

driver-DelimitedFileImport

This component is a history file import descriptor used to create a history based upon data in a (local) delimited text file, such as a comma-separated-values (CSV) or tab-delimited values file (you can configure the delimiter to use). To view Import descriptors expand **FileNetwork→FileDevice→HistoryNetworkExt** (Histories extension).

This import descriptor is similar to the Excel CSV file import descriptor, with the exceptions that the delimited file import descriptor uses a dumb-string tokenizer to parse each line of the specified file and separates table columns based solely on the specified delimiter. You should import only non-complex CSV file or non-CSV delimited file (such as tab-delimited, for example) with the delimited file import descriptor. For any CSV file created by Microsoft Excel, use the Excel CSV file import descriptor instead.

Figure 36 Delimited File Import properties



To create this component, expand **Config→Drivers→NiagaraNetwork**, right-click **Histories**, click **Views** > **Delimited File Import Manager**, click **New**, select **DelimitedFileImport**, click **OK**, configure the import descriptor and click **OK**.

To access these properties after creating an import descriptor, expand **Config→Drivers→NiagaraNetwork→Histories** and double-click **DelimitedFileImport**

In addition to the standard properties (Status, Enabled and Fault Cause), these properties are unique to this component.

Property	Value	Description

State	read-only	A frozen property on a component, indicates the current state of operations.
		`Idle` (default)
		`Pending`
		`In Progress`

Execution Time	additional properties	Configures a time trigger that controls when to perform the function.

Property	Value	Description
		Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the timestamp of the last successful history export.
Last Failure	read-only	Reports the last time the system failed to perform this function. Refer to **Fault Cause** for details.
History Id	text (defaults to:/ `stationname`/ `historyname`, where `stationname` is the name of the Supervisor station and `historyname` is a discovered name)	Specifies the history station and name. If the **Discovered** pane displays the station name as the caret symbol (^), the history name reflects the source history name. Leave both properties at their default values or edit the second (<`historyName`>) property only.
On Demand Poll Enabled	`true` (default) or `false`	Determines user control over polling. `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
On Demand Poll Frequency	drop-down list	References the On Demand Poll Scheduler rates under the **NiagaraNetwork**'s History Policies container slot. Configures how often this type of poll occurs.
Config Overrides, capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Config Overrides, fullPolicy	drop-down list (defaults to `Roll`)	Defines what happens if **Capacity** is set to **Record Count** and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **Capacity** is `Unlimited`.
File	File Chooser	Identifies the local delimited file to import using standard ORD syntax and an absolute file ORD path. If the parent FileDevice has a non-null **Base Ord** property (specifying a directory), you can type in a file name or file path relative to that directory, using the following ORD syntax: `file:fileName` or `filePath/toFileName`.

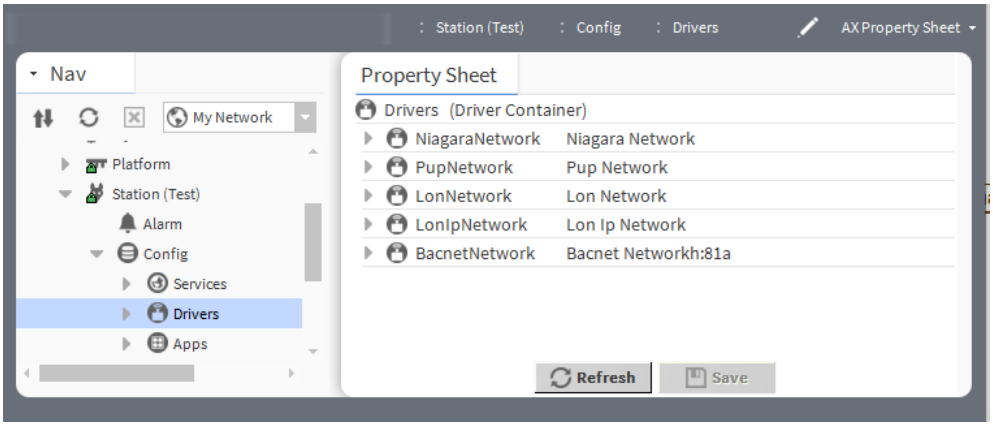
Property	Value	Description
Full Import On Execute	`true` or `false` (default)	Controls how much of history to refresh on each import. `True` imports the entire history on each import. `False` appends only new data to the history on each import.
Row Start	number (defaults to zero)	Specifies the line in the file from which to begin importing data. This value is zero-based, meaning that a value of 1 skips the first line (usually column headers) and begins importing with the second line.
Row End	number (defaults to the end of the file)	Specifies on which line in the file to stop importing data.
Delimiter	text character	Specifies the text character the file uses to separate columns.
Timestamp Column Index	number	Defines a left-to-right index (zero-based) to identify the column in the file from which to import the timestamp. For example: if the timestamp is in the first column, this value is zero (0). If it is in the second column, this value is 1; and so on. This value is required.
Timestamp Format	drop-down list	Specifies a string to define how timestamps are formatted in the file. After selecting an option, you can further edit the string, if necessary. Editing may be necessary if you execute and the descriptor is in fault. This value is required.
Value Column Index	number	Defines a left-to-right index (zero based) to identify the column in the file from which to import the data value. For example, if the second column contains the data value, this value is 1; if the fifth column, this value is 4; and so on. This value is required.
Value Format	drop-down list (defaults to `Numeric Type`)	Specifies the data type of history record to create: `Numeric Type` identifies a numeric history. `String Type` identifies a string history. `Boolean Type` identifies a two-state history.
Status Column Index	number	Defines the left-to-right index (zero-based) to identify the column in the file used to import the status, if available. The imported status value must match the status bits implementation in encoded-integer format, which uses a decimal-equivalent value either singly or, in the case of non-zero values, in combinations: • 0 - ok • 1 - disabled • 2 - fault • 4 - down • 8 - alarm • 16 - stale

Property	Value	Description
		• 32 - overridden • 64 - null • 128 - unackedAlarm This value is optional.
Identifier Column Index	drop–down list (defaults to `None`)	Defines the left-to-right index (zero-based) to identify the column in the file used to filter rows for inclusion in combination with the **`Identifier Pattern`** property value). The default value provides no row filtering. This value is optional.
Identifier Pattern		Specifies the text string in the **`Identifier Column Index`** that the import searches for in all rows, where the import job imports any row with the matching text string. Wildcard characters (*) are supported. This value is optional.

driver-DriverContainer

This component, by convention, contains all networks in a station database. The **Driver Manager** is its primary view.

Figure 37 Drivers container properties



As a container, this property sheet has no unique properties. The networks in the screen capture show examples of the components this folder contains.

driver-ExcelCsvFileImport

This component is a history file import descriptor used to create a history based upon data in any (local) comma-separated-values (CSV) text file created by Microsoft Excel. These history file import descriptors reside under the HistoryNetworkExt (`Histories` extension) of a FileDevice in a FileNetwork. You use the **DelimitedFileImportManager** view of the `Histories` extension to add history file import descriptors.

NOTE: This import descriptor is similar to the DelimitedFileImport descriptor, but assumes CSV data specifically created by Microsoft Excel (it lacks the Delimiter property). This allows complex CSV-delimited data to be successfully imported, using the special rules of Excel CSV generated files. For any other type of delimited data (for example, tab-delimited or pipe-delimited), use the DelimitedFileImport descriptor instead.

This import descriptor has properties common among all history import descriptors, such as Name, History Id, and so on. See History Import Manager. See the next section Properties of history file import descriptors for other configuration properties.

Properties of history file import descriptors

History file import descriptors (DelimitedFileImport, ExcelCsvFileImport) have the following set of configuration properties, which appear in the **New** and **Edit** dialogs for the descriptors:

Property	Value	Description
Value Facets	additional properties	Specifies the units with which to display values imported from the delimited file. On the import descriptor's property sheet, this property is found under **Config Overrides**.
Time Zone	drop-down list	Specifies the time zone for the imported history. On the import descriptor's property sheet, this is property is found under **Config Overrides**.
File	File Chooser	Identifies the local delimited file to import using standard ORD syntax and an absolute file ORD path. If the parent FileDevice has a non-null **`Base Ord`** property (specifying a directory), you can type in a file name or file path relative to that directory, using the following ORD syntax: `file:fileName` or `filePath/toFileName`.
Full Import On Execute	`true` or `false` (default)	Controls how much of history to refresh on each import. Enabled imports the entire history on each import. Disabled appends only new data to the history on each import.
Row Start	number (defaults to zero)	Specifies the line in the file from which to begin importing data. This value is zero-based, meaning that a value of 1 skips the first line (usually column headers) and begins importing with the second line.
Row End	number (defaults to the end of the file)	Specifies on which line in the file to stop importing data.
Delimiter	text character	Specifies the text character the file uses to separate columns.
Timestamp Column Index	number	Defines a left-to-right index (zero-based) to identify the column in the file from which to import the timestamp. For example: if the timestamp is in the first column, this value is zero (0). If it is in the second column, this value is 1; and so on. This value is required.
Timestamp Format	drop-down list	Specifies a string to define how timestamps are formatted in the file. After selecting an option, you can further edit the string, if necessary. Editing may be necessary if you execute and the descriptor is in fault. This value is required.
Value Column Index	number	Defines a left-to-right index (zero based) to identify the column in the file from which to import the data value. For example, if the second column contains the data value, this value is 1; if the fifth column, this value is 4; and so on. This value is required.

Property	Value	Description
Value Format	drop-down list (defaults to `Numeric Type`)	Specifies the data type of history record to create: `Numeric Type` identifies a numeric history. `String Type` identifies a string history. `Boolean Type` identifies a two-state history.
Status Column Index	number	Defines the left-to-right index (zero-based) to identify the column in the file used to import the status, if available. The imported status value must match the Niagara status bits implementation in encoded-integer format, which uses a decimal-equivalent value either singly or, in the case of non-zero values, in combinations: • 0 - ok • 1 - disabled • 2 - fault • 4 - down • 8 - alarm • 16 - stale • 32 - overridden • 64 - null • 128 - unackedAlarm This value is optional.
Identifier Column Index	drop–down list (defaults to `None`)	Defines the left-to-right index (zero-based) to identify the column in the file used to filter rows for inclusion in combination with the **`Identifier Pattern`** property value). The default value provides no row filtering. This value is optional.
Identifier Pattern	*	Specifies the text string in the **`Identifier Column Index`** that the import searches for in all rows, where the import job imports any row with the matching text string. Wildcard characters (*) are supported. This value is optional.

driver-FileDevice

FileDevice is the device level container to import local delimited-type files (such as CSV) as histories. It resides under the **FileNetwork**. It has common device properties and Health and Alarm Source Info slots.

The**FileDevice**'s only device extension, and most important child slot, is the**FileHistoryDeviceExt** (Histories), under which you add file import descriptors. The sole configuration property of importance is **`Base Ord`**, where you can select a specific local directory (use the drop-down control beside the folder icon, choose **Directory Chooser**). This configures relative file ord values in the **`File`** property of the history file import descriptors (under its child **Histories** extension). Otherwise, using the default null Base Ord, you specify absolute file ord values in the child descriptors.

NOTE: A **FileNetwork** is configured with only a single **FileDevice**, with a default null value **`Base Ord`** property. However, to impose some logical grouping, you can create multiple **FileDevices**, and use the **`Base Ord`** scheme (with different directory ords) to separate import descriptors by source.

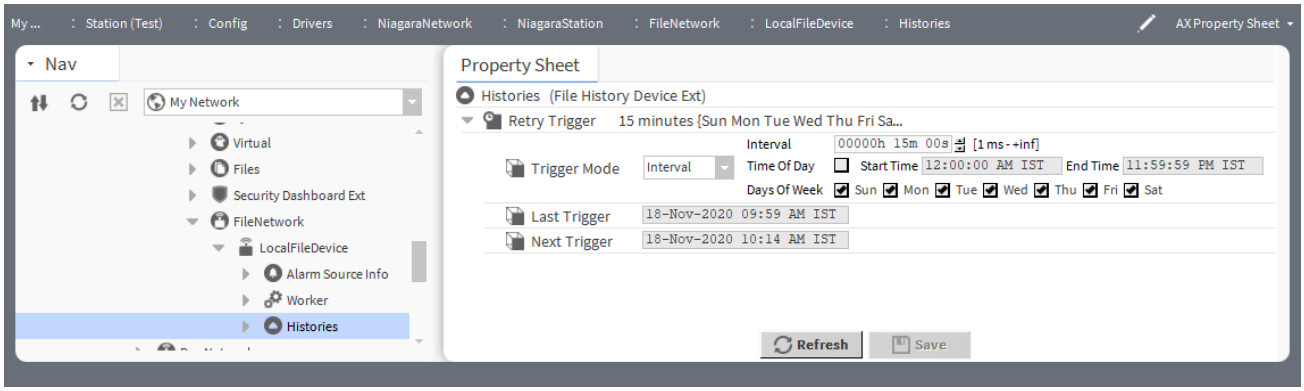
Unlike many field bus drivers, a **FileNetwork** contains no frozen **LocalFileDevice**. Its **FileDevice** named **LocalFileDevice** is a standard **FileDevice** component.

driver-FileHistoryDeviceExt

This component is the only device extension under a **FileNetwork→FileDevice**.

The default view is the **Delimited File Import Manager** with which to add new (or edit existing) file import descriptors. Each descriptor adds the history to the local history space.

Figure 38 FileHistroyDeviceExt properties



To view these properties, expand **Config→Drivers→FileNetwork→FileDevice** right–click **Histories**, click **Views→AX Property Sheet** and expand **Retry Trigger**.

Property	Value	Description
Retry Trigger	additional properties	Defines how frequently to attempt a failed operation again. This continues until successful execution occurs. Appears in the Nav tree but not in any manager view and is unique in that it requires no linking of its output for operation. *Getting Started with Niagara* documents **`Retry Trigger`** properties.

driver-FileHistoryWorker

This component manages the queue and thread processing for history file import descriptors. It is a frozen slot of the **FileDevice**, and requires no configuration.

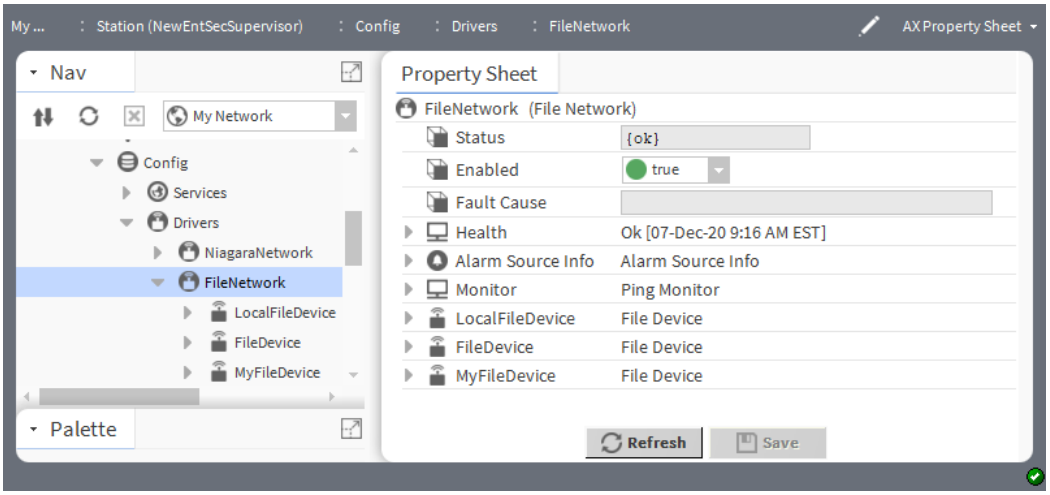
driver-FileNetwork

This **`driver`** module component acts as the network level container for one or more local, imported, delimited-type files, such as CSV files that contain history data. Often, a **FileNetwork** contains only a single **FileDevice**. Unlike a field in a real bus network, the standard driver architecture (network: device: device extensions) provides no real hardware equivalency, this is simply a modeling convention.

Although the **FileDeviceManager** is its primary view, the view used most often under a **FileNetwork** is the **Delimited File Import Manager** view of the **Histories** extension of a **FileDevice**.

A **FileDevice** has none of the other standard device extensions (**Points**, **Schedules**, or **Alarms**).

Figure 39 FileNetwork properties



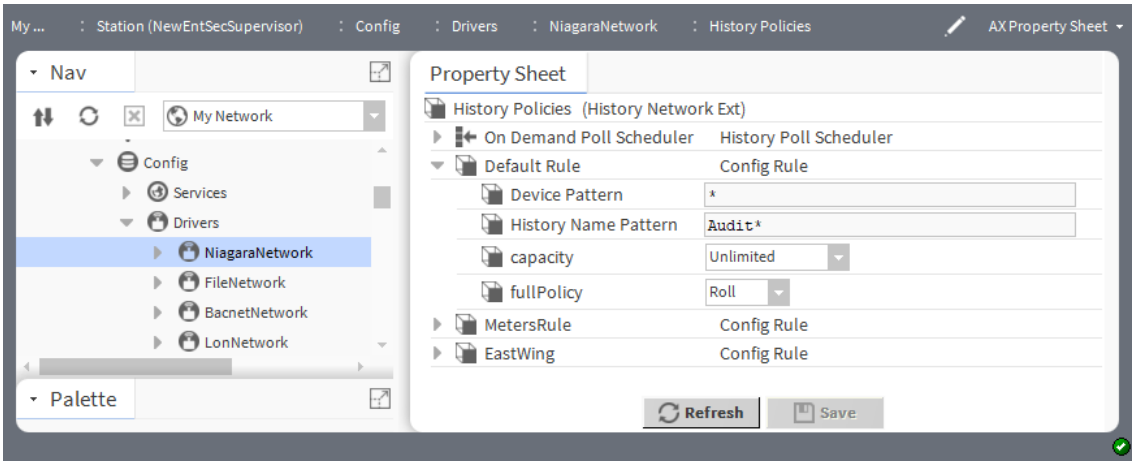
To access these properties, expand **Config→Drivers**, right-click **FileNetwork** and click **Views→AX Property Sheet**.

In addition to the FileDevice extensions, this Property Sheet provides only standard properties (Status, Enabled, Fault Cause, Health, Alarm Source Info and Monitor). These properties are documented elsewhere.

driver-HistoryNetworkExt

This component (**History Policies**) configures history import polling properties and the configuration rule used to export histories from a remote controller station to a local Supervisor station.

Figure 40 History Network Ext properties



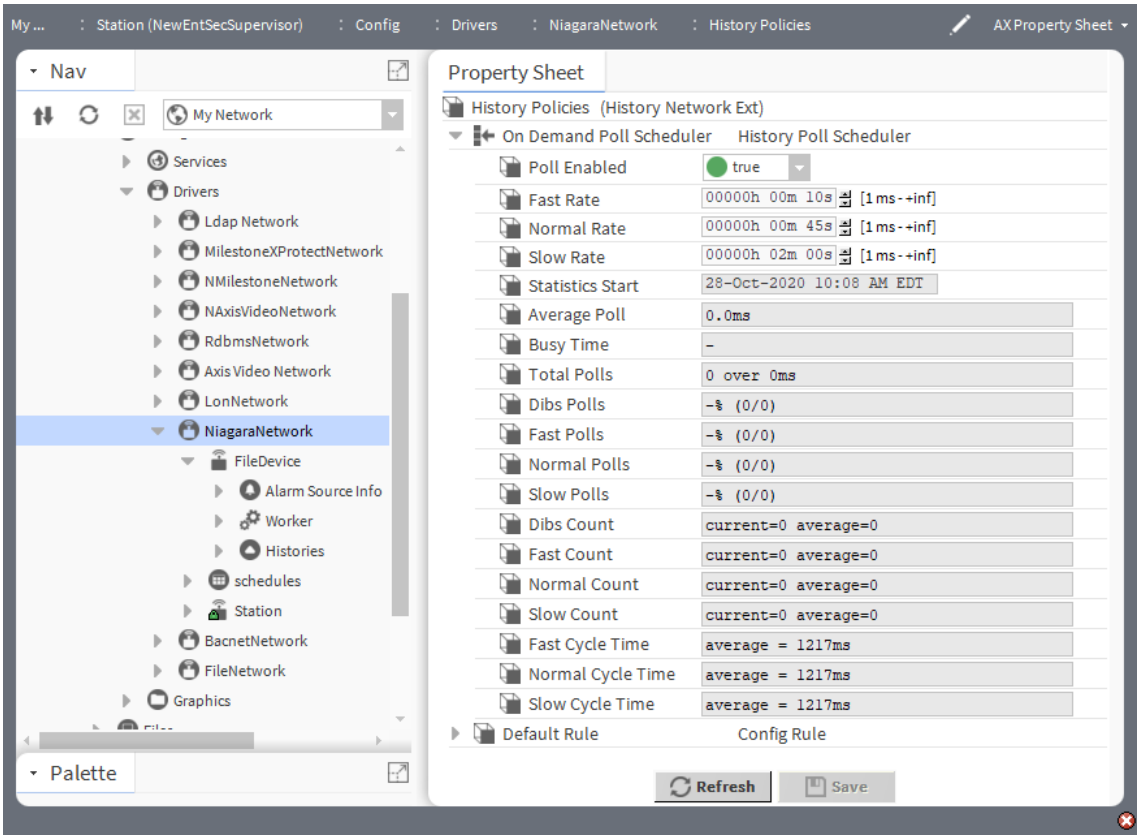
To access these properties, expand **Config→Drivers**, right-click **NiagaraNetwork** and click **Views→AX Property Sheet** and click **History Policies**.

Property	Value	Description
On Demand Poll Scheduler	additional properties	Configures the polling rates used in the on-demand polling of imported histories. The topic, **driver-HistoryPollScheduler** documents these properties.
Default Rule	additional properties	Selects the stations with histories to export and the history names to export from the selected stations. The **driver-ConfigRule** component topic defines these properties. For details, refer to the *Niagara Histories Guide* By default, the parent container **HistoryNetworkExt** (**History Policies**) contains only the single `Default Rule`, which configures all exported histories with unlimited capacity. The system applies configuration rules only when it creates the history, that is, the first time it archives the history in the receiving station. Changing a rule has no effect on existing histories. The topic, "driver-ConfigRule" documents these properties.

driver-HistoryPollScheduler

This component (**On Demand Poll Scheduler**) specifies the available network-level polling rates for imported histories, which the station uses in on-demand polling of an imported history. Clicking the **Live Updates** button on a **History Chart** or **History Table** view triggers such polling.

Figure 41 History Network Ext properties



To display these properties, expand **Config→Drivers**, right-click the **NiagaraNetwork**, click **Views→AX Property Sheet**, expand **History Policies** and double-click **On Demand Poll Scheduler**.

Property	Value	Description
Poll Enabled	`true` (default) `false`	Enables and disables polling. `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
Fast Rate	hours minutes seconds (defaults to 10 seconds)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.
Normal Rate	hours minutes seconds (defaults to 45 seconds)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.
Slow Rate	hours minutes seconds (defaults to two minutes)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.
Statistics Start	read-only	Reports the last time statistics were reset.
Average Poll	read-only	Reports the average time spent in each poll.

Property	Value	Description
Busy Time	read-only	Reports the percentage of the time that the station was busy doing polls.
Total Polls	number of polls over the number of milliseconds	Reports the total number of polls conducted and the time spent waiting for polls to execute.
Dlbs Polls	read-only	Reports the current and average number of components in the DIBS stack. (DIBS stands for Distributed Internet Backup System).
Fast Polls	read-only	Reports the total number of polls made processing the fast queue.
Normal Polls	read-only	Reports the total number of polls made processing the normal queue.
Slow Polls	read-only	Reports the total number of polls made processing the slow queue.
Dlbs Counts	read-only	Reports the percentage and ratio of the number of DIBS polls versus total polls made processing the dibs stack.
Fast Count	read-only	Reports the current and average number of components in the fast queue.
Normal Count	read-only	Reports the current and average number of components in the normal queue.
Slow Count	read-only	Reports the current and average number of components in the slow queue.
Fast Cycle Time	read-only	Reports the average cycle time for the fast queue.
Normal Cycle Time	read-only	Reports the average cycle time for the normal queue.
Slow Cycle Time	read-only	Reports the average cycle time for the slow queue.

driver-PingMonitor

This component configures the ping mechanism. The driver network uses the ping monitor to query network and device health by periodically issuing a ping call to the object. The ping monitor provides built-in support to generate alarms when objects that can receive a ping are down. The **Ping Monitor** is available in the **Property Sheet** of most networks as **Monitor**. These properties contain ping and alarm-related parameters that are common to all Niagara video components.

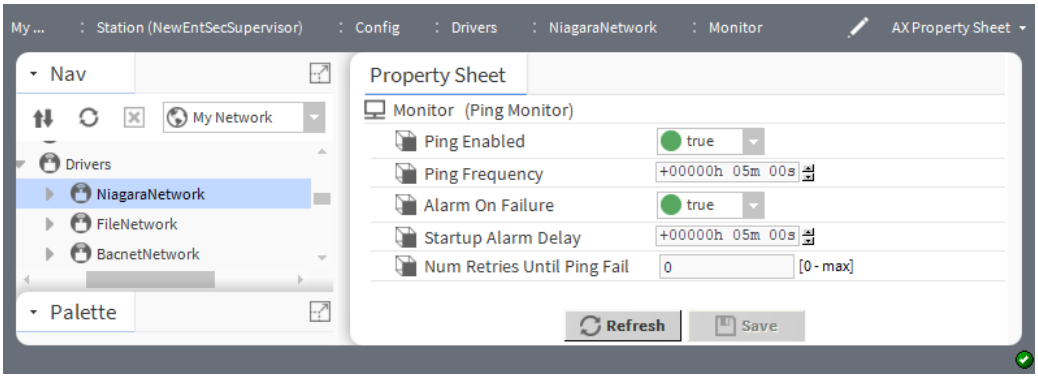
If a device does not respond to the monitor ping, the monitor marks it with a down status and suspends normal communication routines to the device. Upon the next successful monitor ping, device status returns to `{ok}` and normal communication routines resume.

If a device responds to the monitor ping, device status is `{ok}`, and normal communication routines to it (proxy-point polling, plus reads of device schedules, trends, etc. if supported by the driver) proceed normally. This applies even if the device returns an error response to the ping. Any response indicates that the device is alive.

A successful ping response updates an object's **`Health`** property with the current time. The network ping Monitor only pings the device if the time of last health verification is older than the ping frequency. Therefore, in normal operation with most drivers, the proxy point polling mechanism actually alleviates the need for the monitor ping, provided that the ping frequency is long enough. Also, in most drivers, if a point poll

request receives no response (not even a null) from a device, the monitor immediately notes a ping fail condition without waiting for the monitor ping interval.

Figure 42 Ping Monitor properties



To access ping monitor properties under the **NiagaraNetwork**, expand **Config→Drivers**, right-click **NiagaraNetwork**, click **Views→AX Property Sheet** and click **Monitor**.

Property	Value	Description
Ping Enabled	`true` (default) or `false`	Turns the monitor ping on and off. `true` each network device receives a ping, as needed. `false` no network device receives a ping. Device status remains as recorded the last time this property was `true`. Recommendation: leave **`Ping Enabled`** as `true` in almost all cases.
Ping Frequency	hours, minutes, seconds (defaults to 05m 00s)	Specifies the interval between periodic pings of all devices. Typical default value is every 5 minutes (05m 00s), you can adjust differently if needed.
Alarm On Failure	`true` (default) or `false`	Controls the recording of ping failure alarms. `true` records an alarm in the station's AlarmHistory for each ping-detected device event (down or subsequent up). `false` ignores device down and up events.
Startup Alarm Delay	hours, minutes, seconds	Specifies how long a station waits at startup before generating a device down or up alarm. Applies only if the **Monitor**'s **`Alarm On Failure`** property is `true`.
Num Retries Until Ping Fail	number	Sets up a counter. A ping does not fail until it exhausts this number of tries.

driver-SendTimer

This class file provides support for max and min send time on components implementing the **TimedWrites** interface. The **sendTimer** will identify a SendTimes object in the parent path. The first SendTimes encounter walking up from the immediate parent will be used. If no SendTimes object is found then calls to newChange() will result in an single immediate call to sendWrite() on the parent. The **sendTimer** is available in the driver Module.

driver-SendTimes

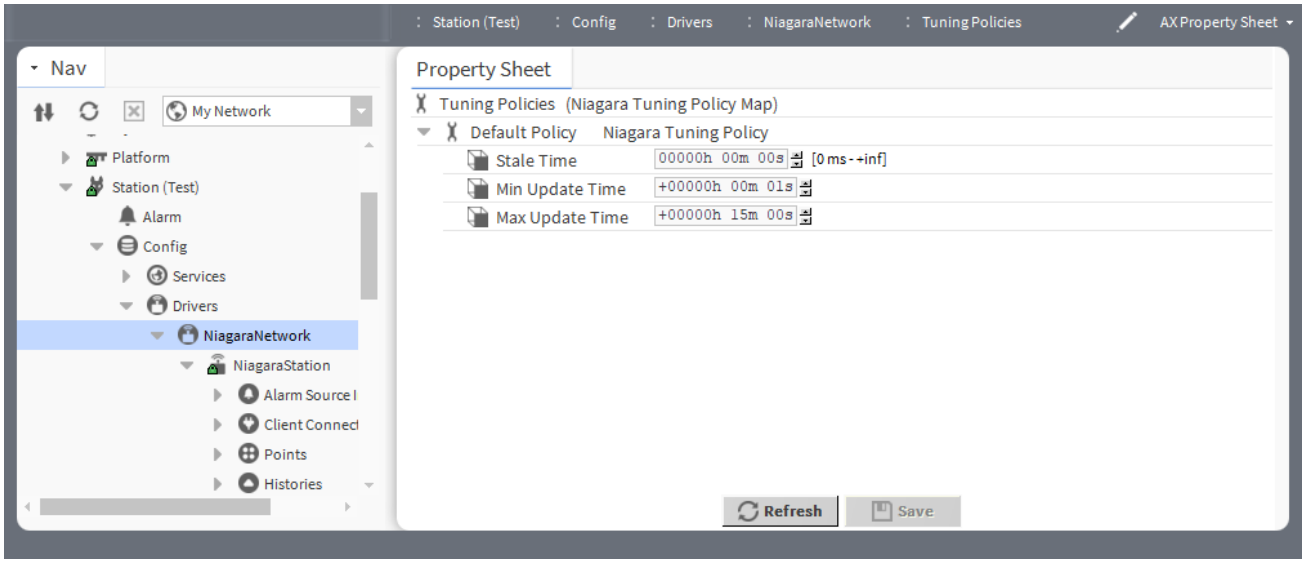
This class file provides the means to configure max and min send times for components implementing the **TimedWrites** interface and containing SendTimer object. The **SendTimes** is available in the driver Module.

driver-TuningPolicy

This component configures the **NiagaraNetwork**'s timing of write requests (for example, writable proxy points) and the acceptable freshness of read requests. Also, can associate with one of 3 Poll Rates in the network's Poll Service.

You can create multiple tuning policies under a driver's **TuningPolicyMap**. You can then assign one or more proxy points to a specific **TuningPolicy**.

Figure 43 Tuning Policy Properties



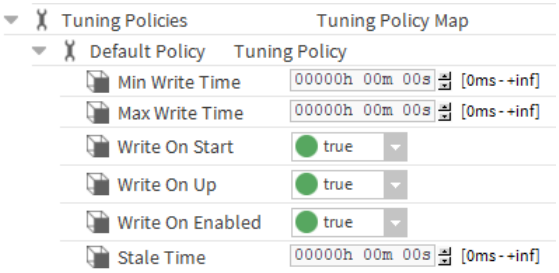
To access this view, expand **Config→Drivers**, right-click **NiagaraNetwork**, click **Views→AX Property Sheet** and click **Tuning Policies**.

Property	Value	Description
Default Policy	additional properties	Specifies additional parameters to be configured.
Stale Time	hours minutes seconds (defaults to 0 seconds)	Defines the period of time without a successful read (indicated by a read status of {ok}) after which a point's value is considered to be too old to be meaningful (stale). A non-zero value causes the point to become stale (status stale) if the configured time elapses without a successful read, indicated by Read Status `{ok}`. The default value (zero) disables the stale timer causing points to become stale immediately when unsubscribed.
Min Update Time	hours minutes seconds (defaults to 01 seconds)	Defines the minimum amount of time between the updates sent by the server to the client.
Max Update Time	hours minutes seconds (defaults to 15 minutes)	Defines the maximum amount of time used by the server to resend the values to the client for subscribed points.

driver-TuningPolicyMap

This component contains one or more TuningPolicy(ies) found in the **Property Sheet** of most network components. You can create multiple tuning policies under a network's **Tuning Policy Map**. You can then assign one or more proxy points to a specific tuning policy.

Figure 44 Network Tuning Policies properties



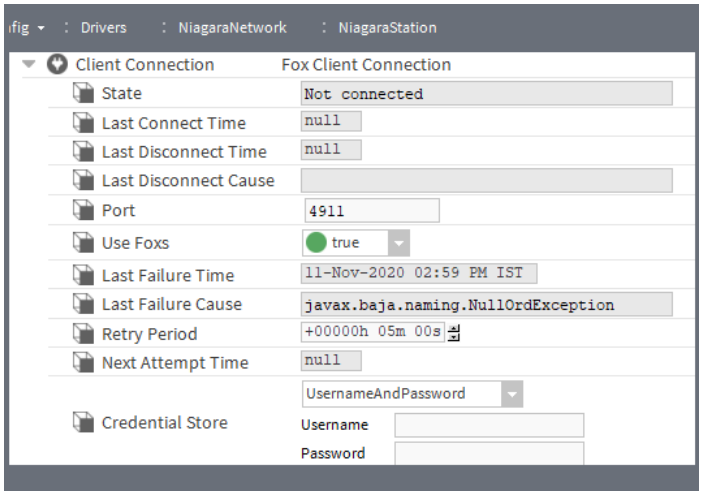
Property	Value	Description
Min Write Time	hours minutes seconds, zero (default) to infinity	Specifies the minimum amount of time allowed between writes to writable proxy points, especially ones that have one or more linked inputs. This provides a way to throttle rapidly changing values so that only the last value is written. The default value (0) disables this rule causing all value changes to attempt to write.
Max Write Time	hours minutes seconds, zero (default) to infinity	Specifies the maximum amount of time to wait before rewriting the value, in case nothing else has triggered a write, to writable proxy points. Any write action resets this timer. The default (0) disables this rule resulting in no timed rewrites.
Write On Start	`true` (default) or `false`	Determines a writable proxy point's behavior when the station starts. `true` initiates a write when the station first reaches a steady state. `false` prevents a write when the station first reaches a steady state. **NOTE:** Consider setting to `false` except for critical proxy points, otherwise large networks may experience write-queue-overflow exceptions.
Write On Up	`true` (default) or `false`	Determines a writable proxy point's behavior when the point and its parent device transition from down to up. `true` initiates a write when a transition from down to up occurs. `false` prevents a write when a transition from down to up occurs.

Property	Value	Description
Write On Enabled	`true` (default) or `false`	Determines a writable proxy point's behavior when the point's status transitions from disabled to normal (enabled). `true` initiates a write when the transition occurs. `false` prevents a write when the transition occurs.
Stale Time	hours minutes seconds, zero (default) to infinity	Defines the period of time without a successful read (indicated by a read status of {ok}) after which a point's value is considered to be too old to be meaningful (stale). A non-zero value causes the point to become stale (status stale) if the configured time elapses without a successful read, indicated by Read Status `{ok}`. The default value (zero) disables the stale timer causing points to become stale immediately when unsubscribed.

fox-FoxClientConnection

This component (**Client Connection**) properties configure a FoxSession connection initiated by this VM (Virtual Machine).A **NiagaraStation** and Workbench use it for a FoxSession.

Figure 45 FoxClientConnections Property Sheet



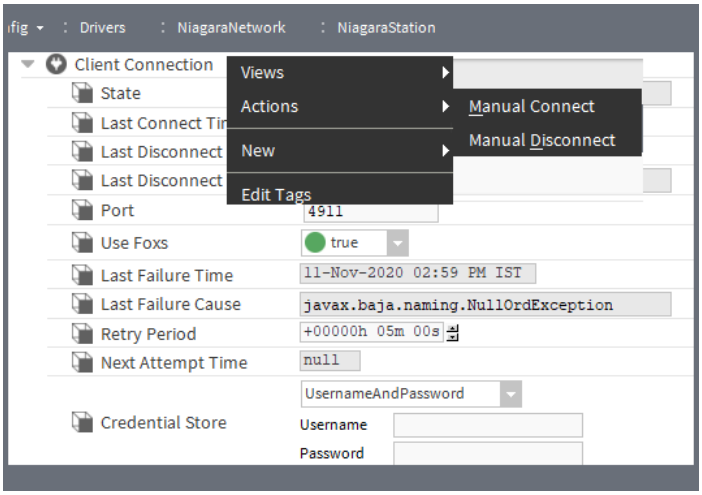
To access this view, expand **Config→Drivers→NiagaraNetwork**, right-click the **NiagaraStation** component in the Nav tree, click **Views→AX Property Sheet** and double-click **Client Connection**.

Property	Value	Description
State	read only	Reports the current state of the Fox connection.
Last Connect Time	read-only	Reports the last connected time of the device to the server.
Last Disconnect Time	read-only	Reports the last disconnected time of the device to the server.
Last Disconnect Cause	read-only	Reports the cause for disconnection of the device to the server.
Port	number	Defines the port number on the controller or computer used to connect to the network.

Property	Value	Description
		If using fox streaming, which uses the station to render the video stream, this port should be different from the station's fox port. If you are not using fox streaming, this port should be the same as the station's fox port.
Use Foxs	`true` (default) or `false`	Turns the use of a secure Fox connection on (`true`) and off (`false`).
Last Failure time	read-only	Reports the time for last failed connection.
Last failed Cause	read-only	Reports the cause for failed connection.
Retry Period	hours minutes seconds	Specifies the period for retry for connection.
Next Attempt Time	read-only	Displays the time for next attempt for connection.
Credential Store	drop-down list	Defines the Username and Password for the connection.

Actions

To access this view, right click **Client Connection**.



This component has two default actions. You can manually force/test a connection using Manual Connect and Manual Disconnect actions.

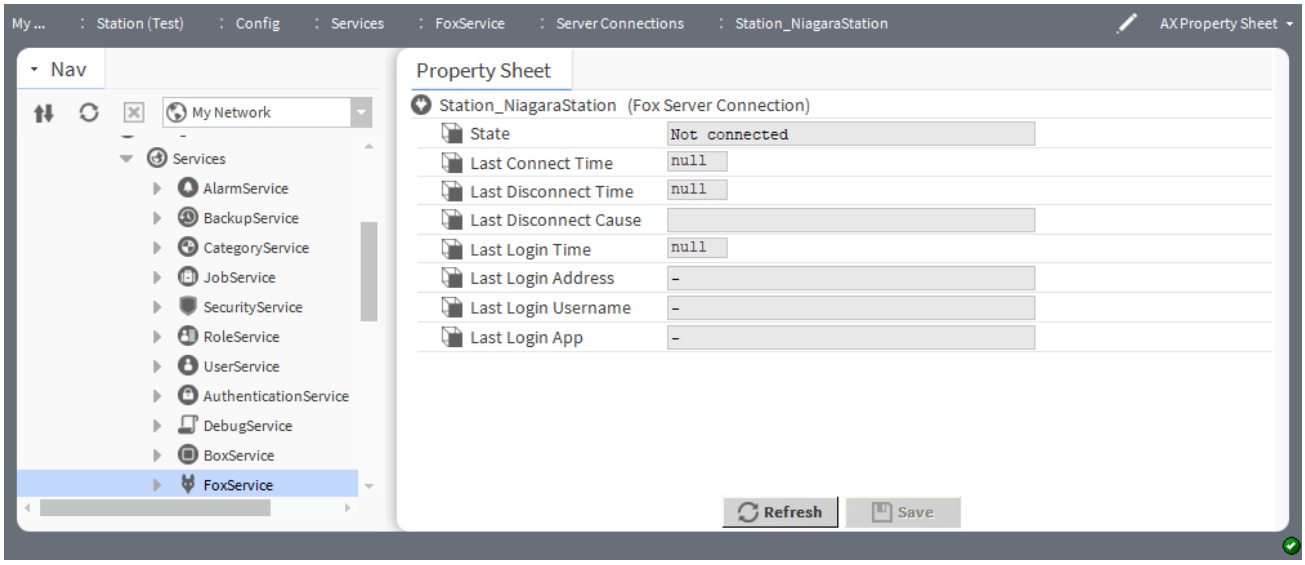
fox-FoxFileSpace

FoxFileSpace maps File: remotely.

fox-FoxServerConnections

This component reports on the Fox (client) session connection to the station's Fox server. It represents user access (vs. access from another station). Child slots provide the connection state as well as various historical data about last connect and disconnect activity. **ServerConnections** is the container slot to store the current server-side **Fox Server Connections** from non-station clients. The primary view is the **Server Connections Summary**. The **ServerConnections** is available in the **`fox`** module.

Figure 46 FoxServerConnections Properties



To access this view, expand **Config→Services**, right-click **FoxService**, click **Views→AX Property Sheet** and click **Server Connections**, double-click the connection.

Property	Value	Description
State	read-only	Indicates the current state of the Fox connection.
Last Connect Time	time-stamp	Displays last successful connection time.
Last Disconnect Time	time-stamp	Displays last disconnection time.
Last Disconnect Cause	read-only	Displays the reason for disconnect.
Last Login Time	time-stamp	Displays the last login time.
Last login Address	Ip Address	Displays the login Ip address.
Last login User Name	read-only	Displays the login user name.
Last login App	read-only	Displays the App fro which the user has logged in.

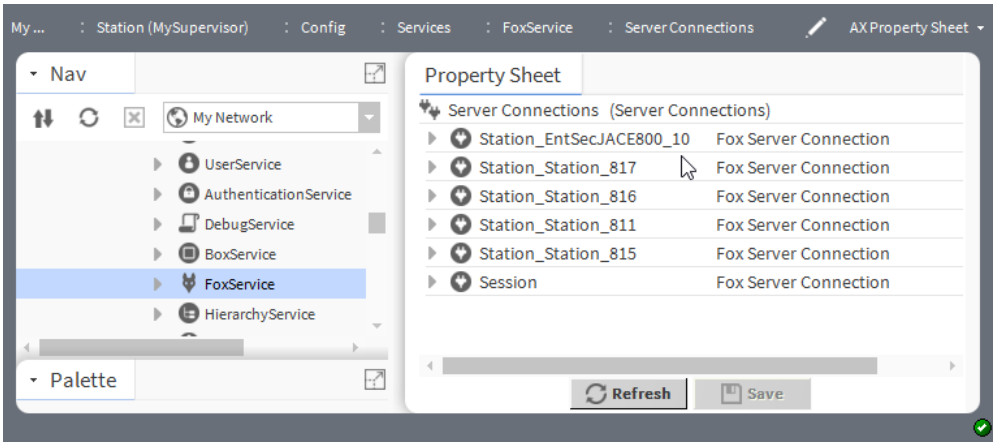
Action

One available action is **Force Disconnect** which disconnects the Fox connection.

Server Connections (fox-ServerConnections)

This component is a container for Fox server connections.

Figure 47 Server Connections properties



To access this property sheet, expand **Config→Services→**, right-click **Server Connections** and click **Views→AX Property Sheet**.

This property sheet has no unique properties of its own.

fox-FoxService

This component is the Baja component wrapper for the FoxServer daemon. It is used within the NiagaraStation to provide basic Fox accessibility. This component is a container slot for Fox protocol settings that affect client connections made to the local station, such as from Workbench or from another station. This specialization of the FoxService maps server connections to the NiagaraStation serverConnection slot. **NiagaraFoxService** typically includes ServerConnections.

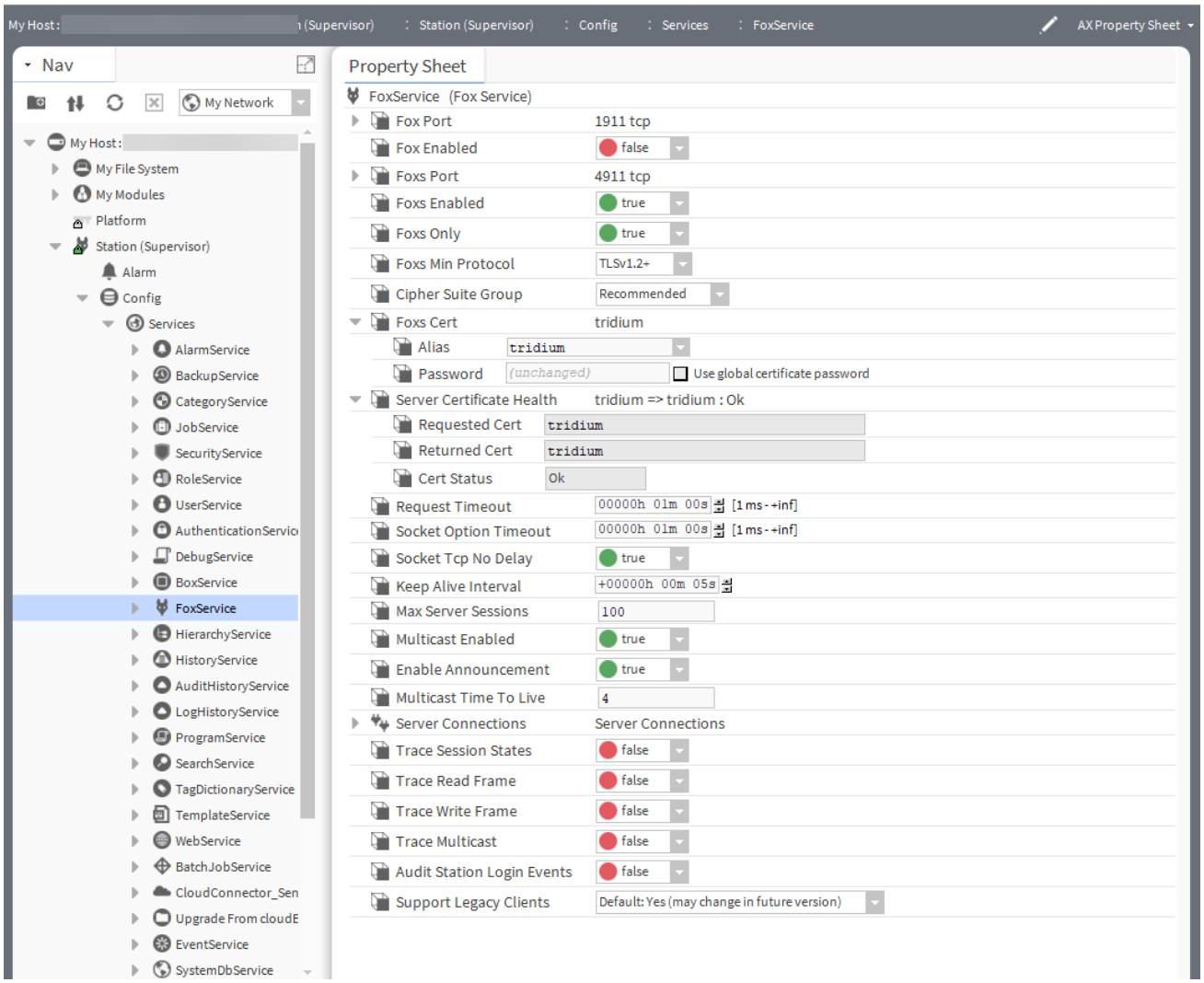
Included are properties for the TCP port number assigned to the Fox server, authentication method used, and various timeout/trace settings. See Fox Service properties for more details.

Authentication is required when establishing any Fox connection to/from the station:

- If opening a station in Workbench, you must enter a valid station username and password in the station login window (otherwise it does not open).
- If accessing a station in a browser as a user, where you also must enter valid user credentials (log in).
- If adding a NiagaraStation to a station's NiagaraNetwork, you must configure username and password properties under its Client Connection slot (otherwise it remains down). Often, you enter the username and password of a specific service-type user account in that station. You also specify the software port used by that station's Fox server.

  NOTE: Often in a multi-station job, in each station you create a user specifically for station-to-station communications, typically with admin write privileges. This is the service-type account that you reference when you edit a NiagaraStation's Client Connection properties, entering its username and password.

Figure 48 Fox Service Properties



To access these properties, expand **Config→Services**, double-click **FoxService** or right-click **FoxService**, click **Views→Property Sheet**.

Property	Value	Description
Fox Port	additional properties	Specifies the TCP port used by the Fox server. "Server Port (baja-ServerPort" documents the additional properties.
Fox Enabled	`true` or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.). When enabled, **`Http Enabled`** in the **`WebService`** must also be set to `true` (for wbapplet use). When set to `false` the system to ignores attempts to connect using Fox port 1911. If **`Foxs Only`** is enabled, this setting (`false` for **`Fox Enabled`**) is irrelevant.

Property	Value	Description
Foxs Port	additional properties	Specifies the TCP port used by the Fox server. "Server Port (baja-ServerPort" documents the additional properties.
Foxs Enabled	`true` or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Foxs Only	`true` (default) or `false`	Enables (`true`) and disables (`false`) secure communication. If `true`, and **`Fox Enabled`** and **`Foxs Enabled`** are both `true`, the driver redirects the fox connection attempts as Foxs connections. If **`Fox Enabled`** is `false` and **`Foxs Enabled`** is `true`, only Foxs connection attempts work; The driver ignores Fox connection attempts.
Foxs Min Protocol	drop-down list (defaults to `Default Policy`)	Selects the earliest version of the TLS (Transport Layer Security) protocol supported by your network. This is the minimum level of the TLS. Options include versions `TLSv1.0+`, `TLSv1.1+`, `TLSv1.2+`, and `TLSv1.3`. Choosing a higher level provides more security. **NOTE:** As of Niagara 4.13, TLSv1.0 and TLSv1.1 are still supported for backwards compatibility, but it is recommended to use TLSv1.2 and higher. During the handshake, the server and client agree on which protocol to use. You should change this property from the default if your network requires a specific version or if a future vulnerability is found in one of the versions.
Cipher Suite Group	`Recommended` (default) or `Supported`	Controls which cipher suites can be used during TLS negotiation. The default is more secure than the other option (`Supported`) and should be used unless it causes compatibility issues with the client.
Foxs Cert	text (read-only)	Displays the host platform's server certificate that is currently used.
Alias	drop-down list (defaults to `default`)	Specifies the alias of the host platform's server certificate, which the client uses to validate server authenticity. The default identifies a self-signed certificate that is automatically created when you initially log on to the server. It cannot be deleted and should be used for recovery purposes. The default certificate is protected by the global certificate password. If other certificates are in the host platform's key store, you can select them from the drop-down list.
Password	text and check box	As of Niagara 4.13, the server certificate is password-protected by either a unique password or the global certificate password. Prompts the user to provide the user-defined password or the global certificate password associated with the server certificate.

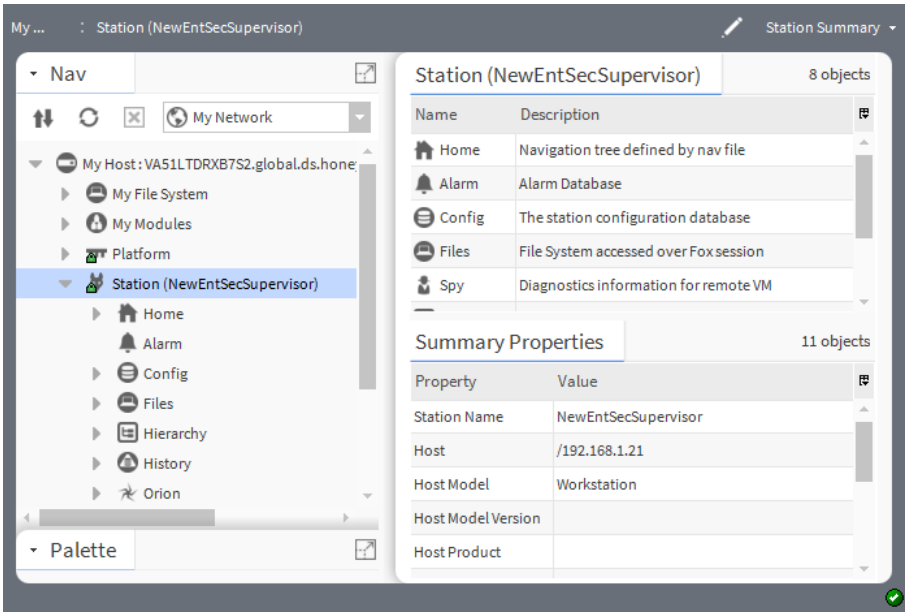
Property	Value	Description
Server Certificate Health	text (read-only)	Displays the alias of the used server certificate and its status (`OK`, `Bad Password`)
Requested Cert	text (read-only)	Displays the certificate's alias that was requested.
Returned Cert	text (read-only)	Displays the actual certificate that is currently used.
Cert Status	text (read-only)	Specifies the status of the requested certificate (`OK`, `Bad Password`)
Request Timeout	hours minutes seconds (defaults to one minute	Defines how long to wait for a response before assuming a connection is dead.
Socket Option Timeout	hours minutes seconds (defaults to one minute)	Defines how long on a socket read before assuming the connection is dead.
Socket Tcp No Delay	`true` (default) or `False`	Disables (`true`) and enables (`false`) Nagle's algorithm, which causes issues with delayed acknowledgements that occurred in TCP socket communications between Fox clients and servers. The default is recommended, which disables Nagle's algorithm. On the Workbench side, a line added to the `system.properties` file can adjust this setting: `niagara.fox.tcpNoDelay=true`.
Keep Alive Interval	hours minutes seconds (defaults to five (5) seconds)	Defines the interval between keep alive messages. The keep alive should be well below the request time-out and socket option time-out.
Max Server Sessions	number (defaults to 100)	Defines the maximum number of Fox/Foxs server connections before additional client connections error with busy.
Multicast Enabled	`true` (default) or `False`	Enables (`true`) and disables (`false`) UDP multicasting initiated by the station. This is necessary for a discovery from this station. This differs from Workbench UDP mulitcast support, which can be disabled via an entry in the Workbench host's `system.properties` file.
Enable Announcement	`true` (default) or `False`.	Enables (`true`) and disables (`false`) support of UDP multicast announcement messages received by the station in support of learn/discovery.
Multicast Time To Live	number (defaults to 4)	Defines the number of hops to make before a multicast message expires.
Server Connections	additional properties	Provides status information about current Workbench client connections to the local station (does not reflect station-to-station Fox connections).
Trace Session States	`true` or `False` (default)	Enables (`true`) and disables (`false`) debug usage for tracing session state changes.
Trace Read Frame	`true` or `False` (default).	Enables (`true`) and disables (`false`) debug usage for dumping frames being read from the wire.
Trace Write Frame	`true` or `False` (default).	Enables (`true`) and disables (`false`) debug usage for dumping frames being written to the wire.

Property	Value	Description
Trace Multicast	`true` or `False` (default).	Enables (`true`) and disables (`false`) debug usage for tracing multicast messaging.
Audit Station Login Events	`true` or `False` (default).	Enables (`true`) and disables (`false`) the auditing of Fox login and logout events for station clients. When set to `true`, increase the capacity for the audit history accordingly.
Support Legacy Clients	drop-down list	Selects legacy client versions to support.

fox-FoxSession

This component represents the Workbench (client) session to a station running on a Niagara host (server), using a Fox connection on a particular port—in other words, a station connection.

Figure 49 Example of a Fox session



You see this as an expandable icon in the Nav tree, followed by the station's (name), for any station opened in the Workbench. To open this view, double-click the Station node in the Nav tree. There is no Property Sheet view for the **fox-ForSession** (station) component.

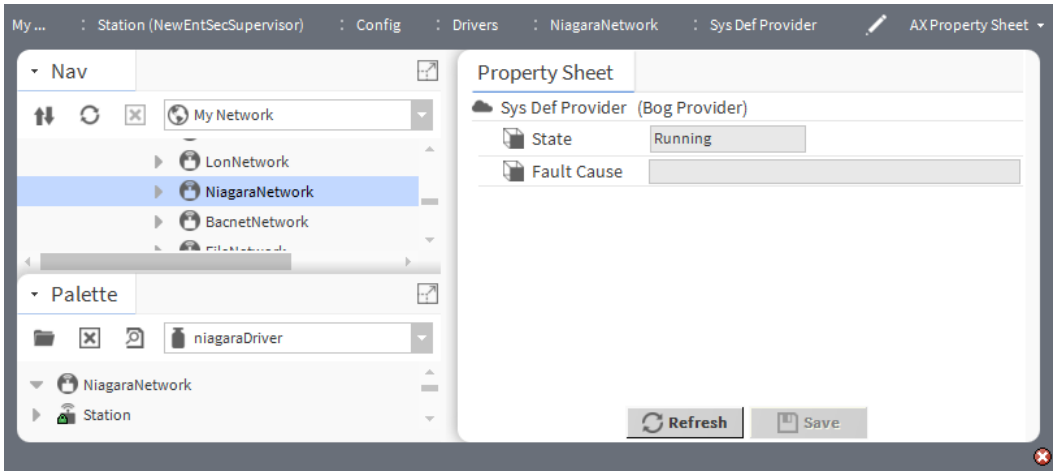
A secure Fox station connection (SSL or TLS) is also possible from Workbench, in which case the station connection icon shows a small padlock.

Also, Workbench provides a right-click **Session Info** command on any active station connection, which produces a popup dialog with details on the current Fox station connection. This is in addition to other right-click commands previously available, such as Disconnect, Close, Spy, Save Station, Backup Station, and so on.

niagaraDriver-BogProvider

This component is a child container property in the **NiagaraNetwork**. The API interacts with this component to query the Sys Def hierarchy, and persists this definition. Storage includes two options to store all Sys Def nodes: either BOG, that is, the station .bog, as the default (using child ProviderStation components) or Orion (in Orion database). Sys Def is chiefly of interest to developers working with the API.

Figure 50 Bog Provider Sys Def properties



To access these properties, expand **Config→Drivers→NiagaraNewtork** and double-click **Sys Def Provider**.

Property	Value	Description
State	read-only	Indicates the current state of the Bog Provider.
Fault Cause	read-only	Reports the reason that the station is in fault.

niagaraDriver-CyclicThreadPoolWorker

The **CyclicThreadPoolWorker** (**Workers**) slot of the NiagaraNetwork allows shared thread pool size tuning for large networks via a Max Threads property. By default, the value of this property is `max` (maximum). If necessary, this can be adjusted.

niagaraDriver-LocalSysDefStation

This component is a child container of a **NiagaraStation**. It reflects common system definition (Sys Def) properties for the local station, which are synchronized with a remote Supervisor station (if the local station is its subordinate). Sys Def properties are chiefly of interest to developers extending the API.

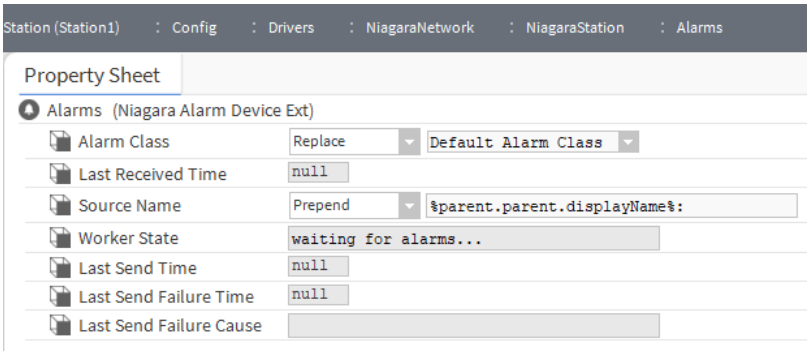
niagaraDriver-NiagaraAlarmDeviceExt

This component (**Alarms** extension) specifies how to map alarms from a station into the current station's own alarm subsystem, plus provide status properties related to alarm sharing.

Only drivers that support local (native) alarming provide this extension under the device component. This includes the **NiagaraNetwork**'s **NiagaraStation**, **BacnetNetwork** and **ObixNetwork**. Unlike some other device extensions, this extension provides no special views, nor does it contain other special components.

The alarms extension under an **ObixClient** device (**ObixDriver**) is modeled differently, where properties specify the oBIX watch interval and URI, and one or more child **ObixAlarmImport** components specify the available alarm feeds from the represented oBIX server. Each alarm feed (**ObixAlarmImport**) has properties to specify the alarm class with which to map into the station's subsystem, as well as status properties. This alarms device extension is unique in its special **Obix Alarm Manager** view, from which you discover, add, and manage alarm feeds.

Figure 51 Alarm Device Ext properties



To access this view under the **NiagaraNetwork**, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click the **Alarms** component in the Nav tree, and click **Views→AX Property Sheet**.

Property	Value	Description
Alarm Class, first drop-down list	drop-down list (defaults to `Use Existing`)	Specifies the alarm routing option for the component. `Replace` provides a selection list of a local alarm classes, from which to select one to use for all alarms received from this device. `Use Existing` routes alarms from this remote station to any matching alarm class, that is, one with an identical name as that in each alarm record. If the program finds no local matching alarm class, it uses the station's default alarm class. `Prepend` adds leading text (as specified) to the incoming alarm class string, then routes it to any local matching alarm class in the station. `Append` adds trailing text (as specified) to the incoming alarm class string, then routes it to any local matching alarm class in the station.
Alarm Class, second property	text field if the first value is `Prepend` or `Append`; drop-down list if the first value is `Replace` (defaults to Medium)	Provides additional information based on the selection made in the drop-down list on the left. The empty text field determines a prefix and suffix for the alarm class. The `Replace` drop-down list selects among: Medium High Low Off Alarm Class
Last Received Time	read-only	Displays the last time the Supervisor station received an alarm from this device.
Source Name	drop-down list (defaults to Prepend with ORD: `%parent.parent.displayName%:`	Identifies the source of the alarm.`Replace` uses the format text entered in this property as alarm source (ignoring whatever incoming alarmSource string is receives). `Use Existing`: Uses only the incoming alarmSource string and ignores the format text entered in this property.

Property	Value	Description
		`Prepend`: T(default) Adds leading text (as specified) to the incoming alarmSource string, where the default format is `%parent.parent.displayName%`. `Append` adds trailing text (as specified) to the incoming alarmSource string. For example, if the format entered is -`%parent.parent.displayName%`.
Worker State	read-only	Reports the state of the alarm system process.
Last Send Time	read-only	Reports when the last local alarm was routed to this device. This is configured under the local station's AlarmService with a corresponding StationRecipient or BacnetDestination component linked to one or more AlarmClass components.
Last Send Time Failure	read-only	Reports when the last local alarm routed to this station could not be sent.
Last Send Failure Cause	read-only	Reports the cause of the failure to rout local alarms to this station.

niagaraDriver-NiagaraEdgeStation

The NiagaraEdgeStation device type is used in a NiagaraNetwork for connections to remote Niagara Edge devices. This component is found in the **niagaraDriver** palette.

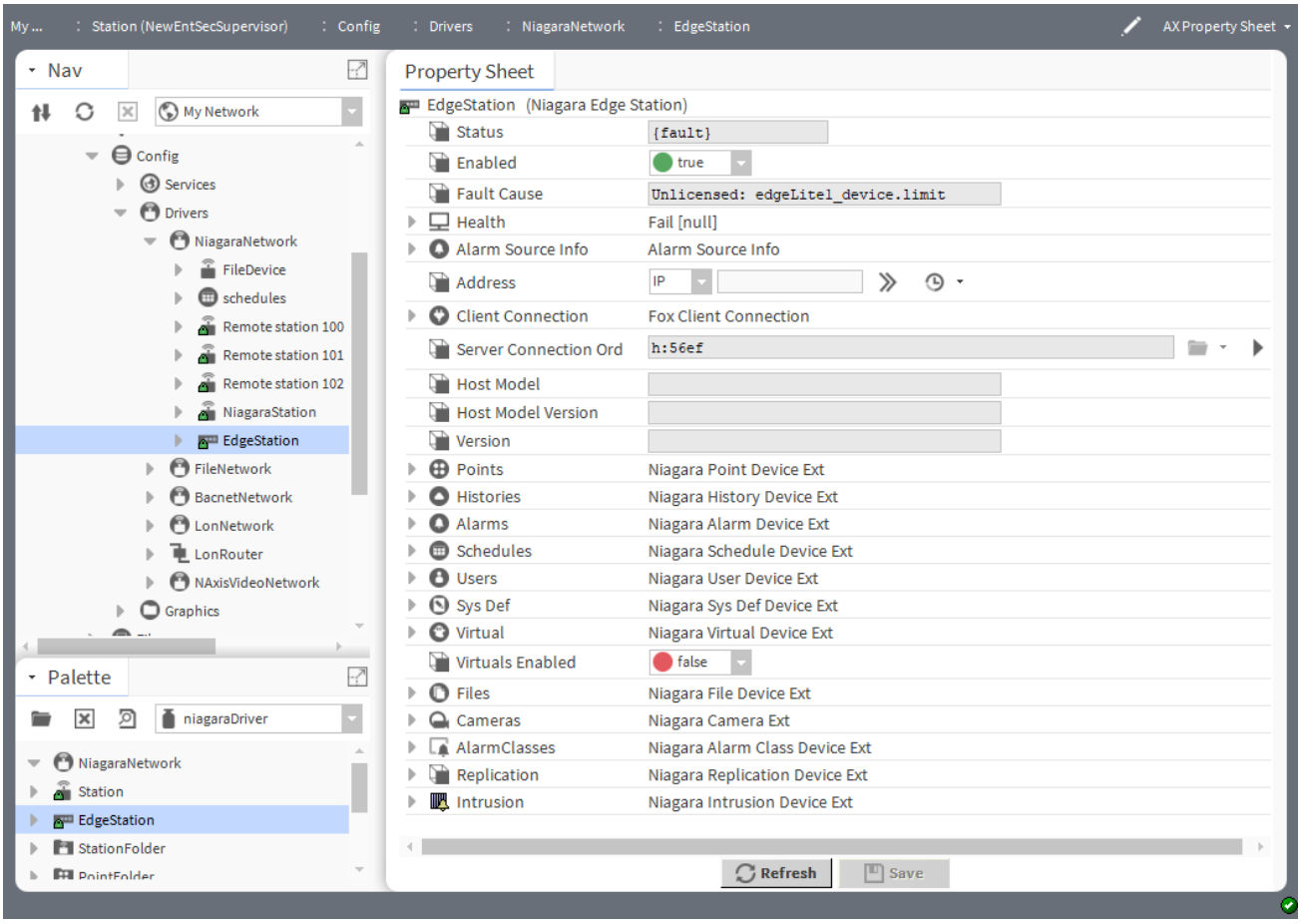
The NiagaraEdgeStation functions in much the same way as the existing NiagaraStation type (supporting the same points, history imports/exports, schedule imports/exports, etc.), the difference is that it is subject to different license limits in a Supervisor, and it will fail to connect to the remote device if it detects that the remote device is not a Niagara Edge device. Therefore, it cannot be used to connect to a remote JACE (those connections should continue to use the existing NiagaraStation type). This feature is subject to license attributes on both the Supervisor and the Niagara Edge station.

For more details on Edge devices, refer to the *Niagara Edge 10 Install and Startup Guide*.

Actions

- Ping — Perform a ping test of the device.
- Force Update Niagara Proxy Points — In Niagara, use this action to force a re-fetch of `n:history` tag information. The action is available on the following components: NiagaraNetwork, NiagaraStation, NiagaraEdgeStation, NiagaraStationFolder, NiagaraPointDeviceExt, and NiagaraPointFolder.

Figure 52 Edge Station properties

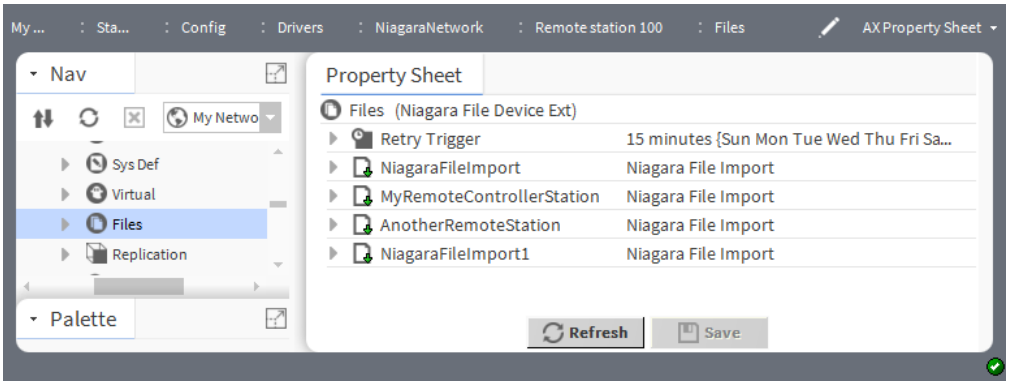


niagaraDriver-NiagaraFileDeviceExt

This device extension (**Files**) contains folders and files to import from the associated remote station. This component is included among the collection of device extensions under every remote **NiagaraStation** component.

The default view for this component is the **Niagara File Manager** in which you create **Niagara File Import** descriptors, to discover, add, and edit properties that specify which files and folders to import.

Figure 53 Niagara File Device Ext properties



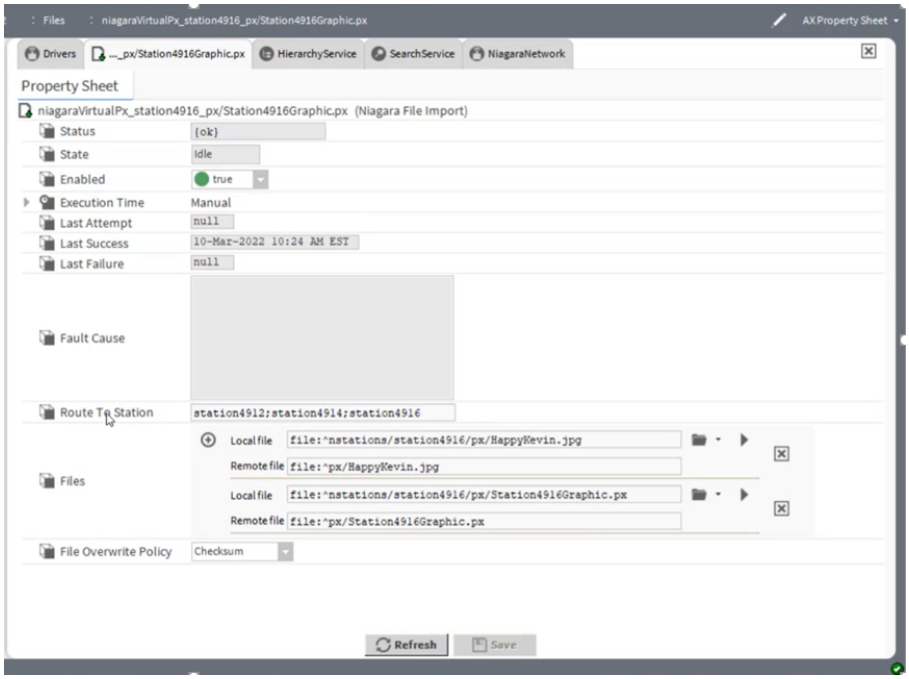
To access this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Files** and click **Views→AX Property Sheet**

Property	Value	Description
Retry Trigger	additional properties	Defines how frequently to attempt a failed operation again. This continues until successful execution occurs. Appears in the Nav tree but not in any manager view and is unique in that it requires no linking of its output for operation. *Getting Started with Niagara* documents **`Retry Trigger`** properties.

niagaraDriver-NiagaraFileImport

This component is an import descriptor that specifies a file or folder of files (including all subfolders and files) to be imported to the local station, sourced from the remote **Station** in a single-tier or multi-tier Niagara system.

Figure 54 Niagara File Import properties



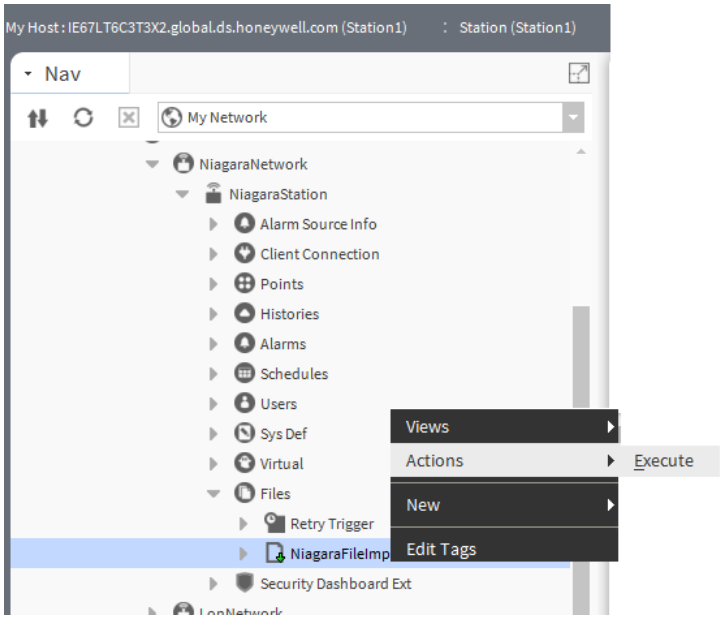
Expand **Config→Drivers→NiagaraNetwork→NiagaraStation**. double-click **Files** and click **Discover**. After the discovery job finds the files, select the ones you wish to add to the Database and click **Add**. Additionally, you can create a new file import descriptor by clicking **New**. Switch to the **Property Sheet** by right-clicking **Files** and clicking **Views→AX Property Sheet**. Then, to open the view above, expand the **NiagaraFileImport** descriptor.

In addition to the standard properties (Status, Enabled and Fault Cause) these properties configure this component.

Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties (**`Trigger Mode`** defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.
Route To Station	text	As of Niagara 4.13 and later, optionally specifies a route of intermediate station names (delimited by semi-colons) to reach the remote, reachable station from which to perform the file import operation. If the field remains empty, the file is imported from the directly connected station (just like the pre-Niagara 4.13 behavior). If a station route is provided, the file is imported from the downstream reachable station (the final station name in the list) and transmitted through all intermediate stations, as indicated by the station route, back up to this station. In order to work, it requires active NiagaraNetwork connections for all intermediate stations specified in the route.
Files	text	Selects the receiving target and sending source file (or directory) pair(s).
File Overwrite Policy	drop-down (defaults to `Checksum`)	Selects the criterion to apply to overwrite existing files upon any import execution: `Checksum` compares a checksum from the file being imported with the checksum from the existing file. If the app finds a difference, it imports the file again and overwrites the existing file. `Last Modified` compares the date and time stamps of the existing and imported files. Any remote files found with a more recent date or time overwrite the existing files.

Action

Figure 55 File Import actions menu



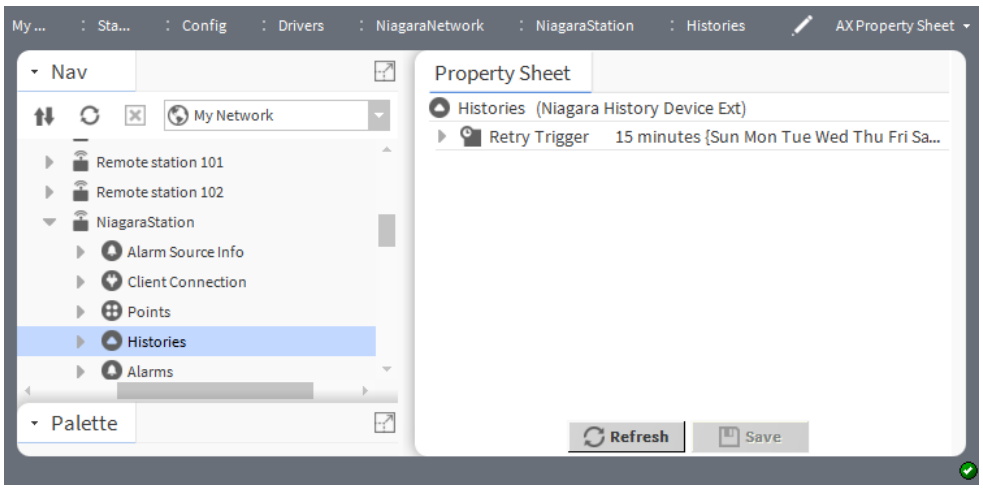
This component has one action, **Execute**. This action imports the selected file(s).

niagaraDriver-NiagaraHistoryDeviceExt

This component is an implementation of **HistoryDeviceExt**. A device's **Histories** extension serves as the parent container for history descriptors—components that specify how history-type collections (log data) are imported or exported. By default, it also contains a **Retry Trigger**.

Only drivers that provide local data logging have this component. This includes the **NiagaraNetwork**'s **NiagaraStation**, **ObixClient**, **R2ObixClient** and **BacnetDevice** (under this device, this extension is named **Trend Logs**). Database device components in any of the **RdbmsNetwork** drivers (**`rdbSqlServer`**, **`rdbMySQL`**, etc.) each have a **Histories** extension.

Figure 56 History Device Ext properties



To access these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Histories** and click **Views→AX Property Sheet**.

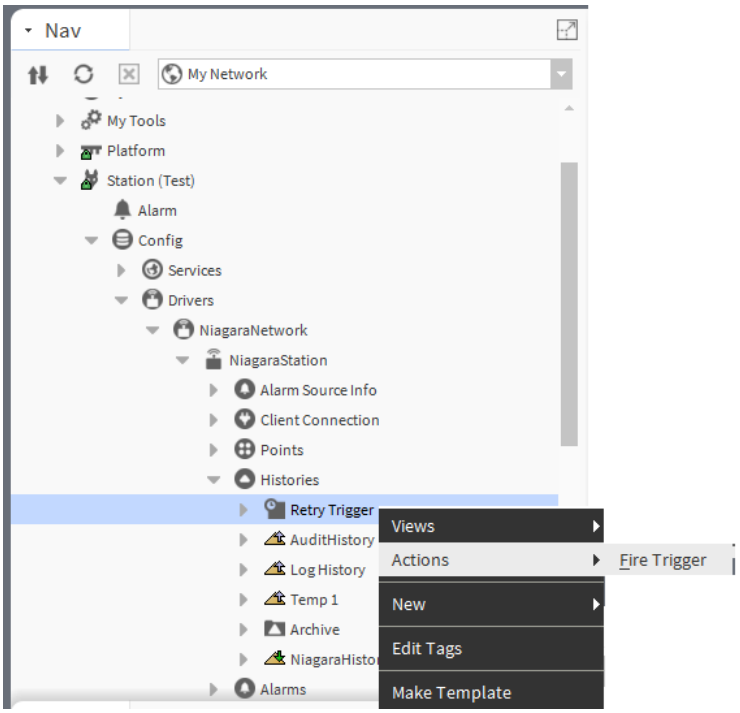
Property	Value	Description
Retry Trigger	additional properties	Defines how frequently to attempt a failed operation again. This continues until successful execution occurs. Appears in the Nav tree but not in any manager view and is unique in that it requires no linking of its output for operation. *Getting Started with Niagara* documents **`Retry Trigger`** properties.

Retry Trigger action

By default, device extensions like Histories, Schedules, and others contain a **Retry Trigger**.

If a descriptor component (that is, a history import descriptor, schedule import descriptor, and so on) fails, the **Retry Trigger** defines subsequent retries that automatically occur at the defined interval (default value is 15 minutes). This continues until a successful execution occurs.

Figure 57 Retry Trigger actions menu

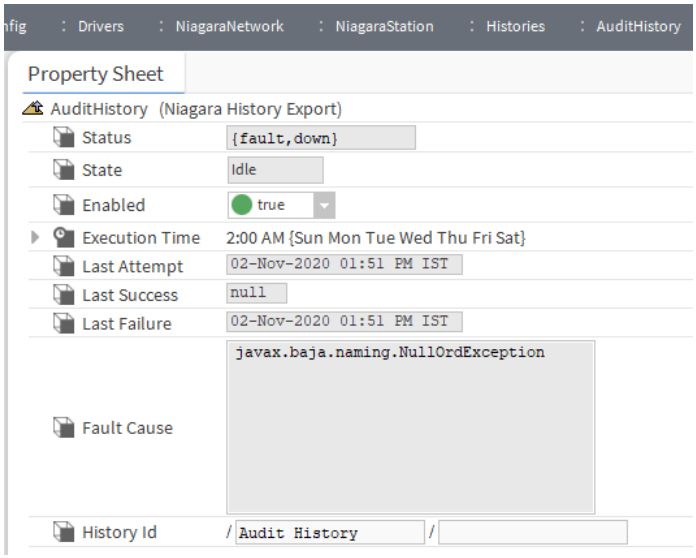


If the histories descriptor fails, right-clicking **Retry Trigger** and click **Actions→Fire Trigger** manually triggers the histories descriptor again.

niagaraDriver-NiagaraHistoryExport

This component defines the export properties for a local history, including current status, and remote history Id. **Niagara History Exports** reside under the **Histories** extension of the **NiagaraStation** in the **Niagara-Network**. For more information, refer to the *Niagara Histories Guide*.

Figure 58 Niagara History Export properties



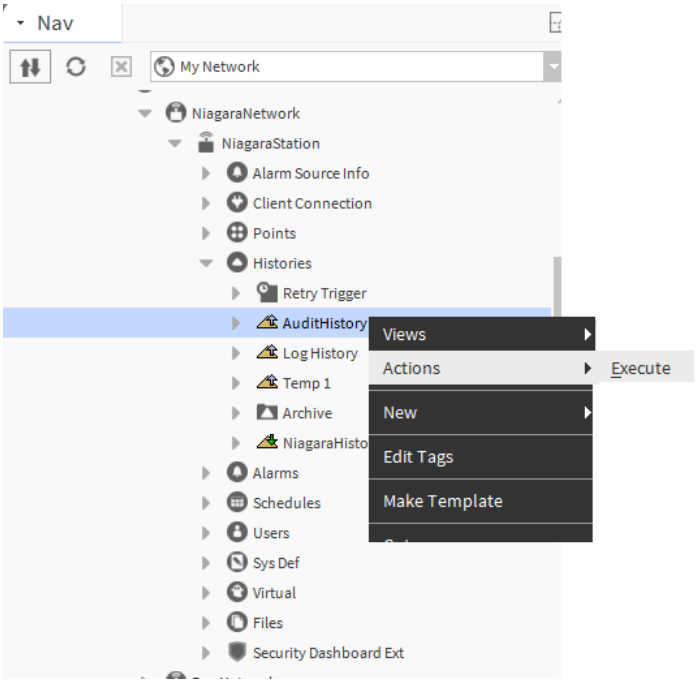
To access this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation→Histories**, right-click the history component in the Nav tree, and click **Views→AX Property Sheet**.

In addition to the standard properties (Status, Enabled and Fault Cause), these properties are unique to this component.

Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties (**`Trigger Mode`** defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.
History Id	text (defaults to:/ `stationname`/ `historyname`, where `stationname` is the name of the Supervisor station and `historyname` is a discovered name)	Specifies the history station and name. If the **Discovered** pane displays the station name as the caret symbol (^), the history name reflects the source history name. Leave both properties at their default values or edit the second (<`historyName`>) property only.

Action

Figure 59 NiagaraHistoryExport actions menu

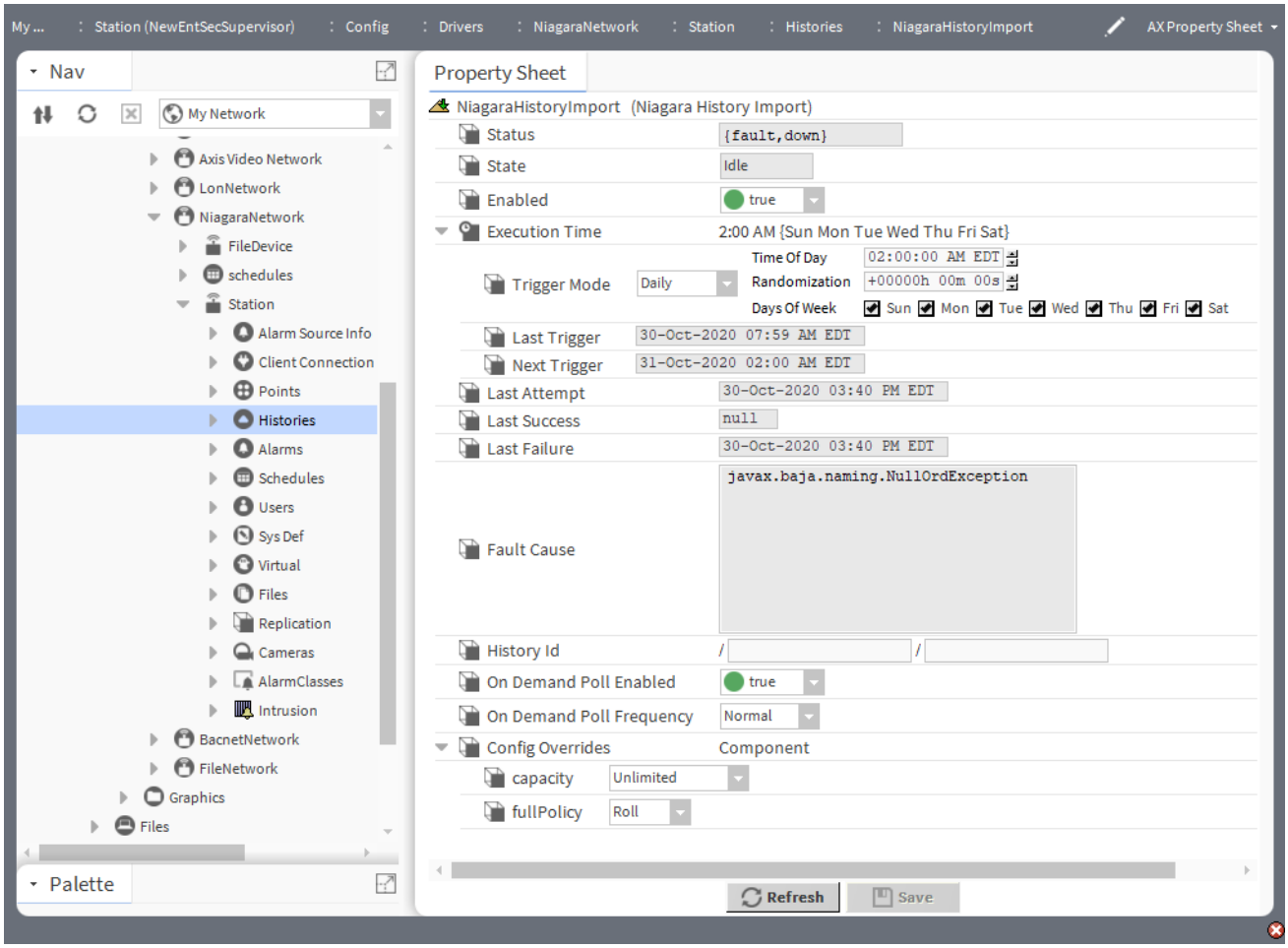


This component has one action, **Executes**, which exports of all selected histories.

niagaraDriver-NiagaraHistoryImport

This component configures import properties for a remote history, including collection (pull) times, current status, local history Id, and config overrides. History imports reside under the **Histories** extension of the **NiagaraStation** in the **NiagaraNetwork**.

Figure 60 Niagara History Import properties



To view these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Histories** and click **Views→AX Property Sheet**.

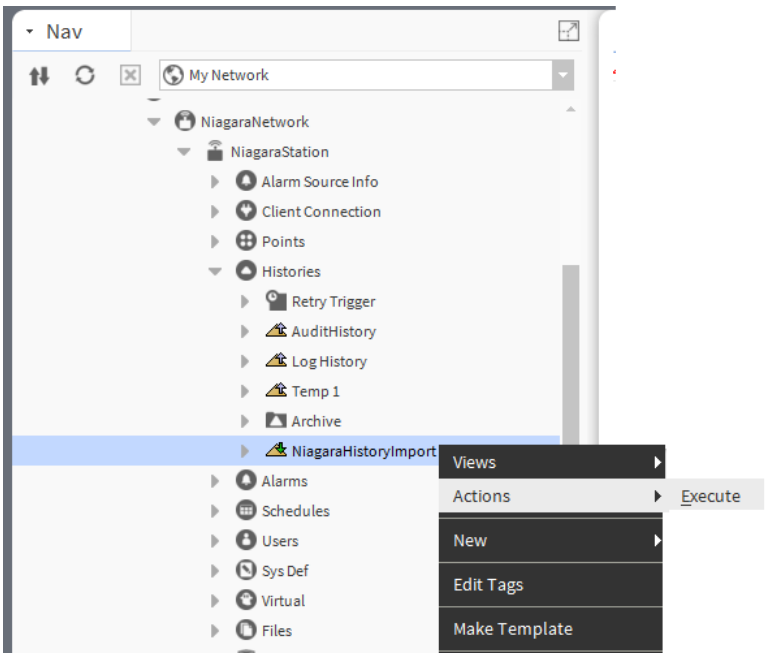
In addition to the standard properties (Status, Enabled and Fault Cause) these properties configure this component.

Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties (**`Trigger Mode`** defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.
On Demand Poll Enabled	`true` (default) or `false`	Enables and disables polling.

Property	Value	Description
		`true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
On Demand Poll Frequency	drop-down list	References the **`On Demand Poll Scheduler`** rates under the **NiagaraNetwork**'s **History Policies** container slot.
Config Overrides, capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Config Overrides, fullpolicy	drop-down list (defaults to `Roll`)	Defines what happens if **`Capacity`** is set to **`Record Count`** and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.

Action

Figure 61 NiagaraHistoryImport actions menu



This component has one action, **Executes**, which imports of all the selected histories.

niagaraDriver-NiagaraNetwork

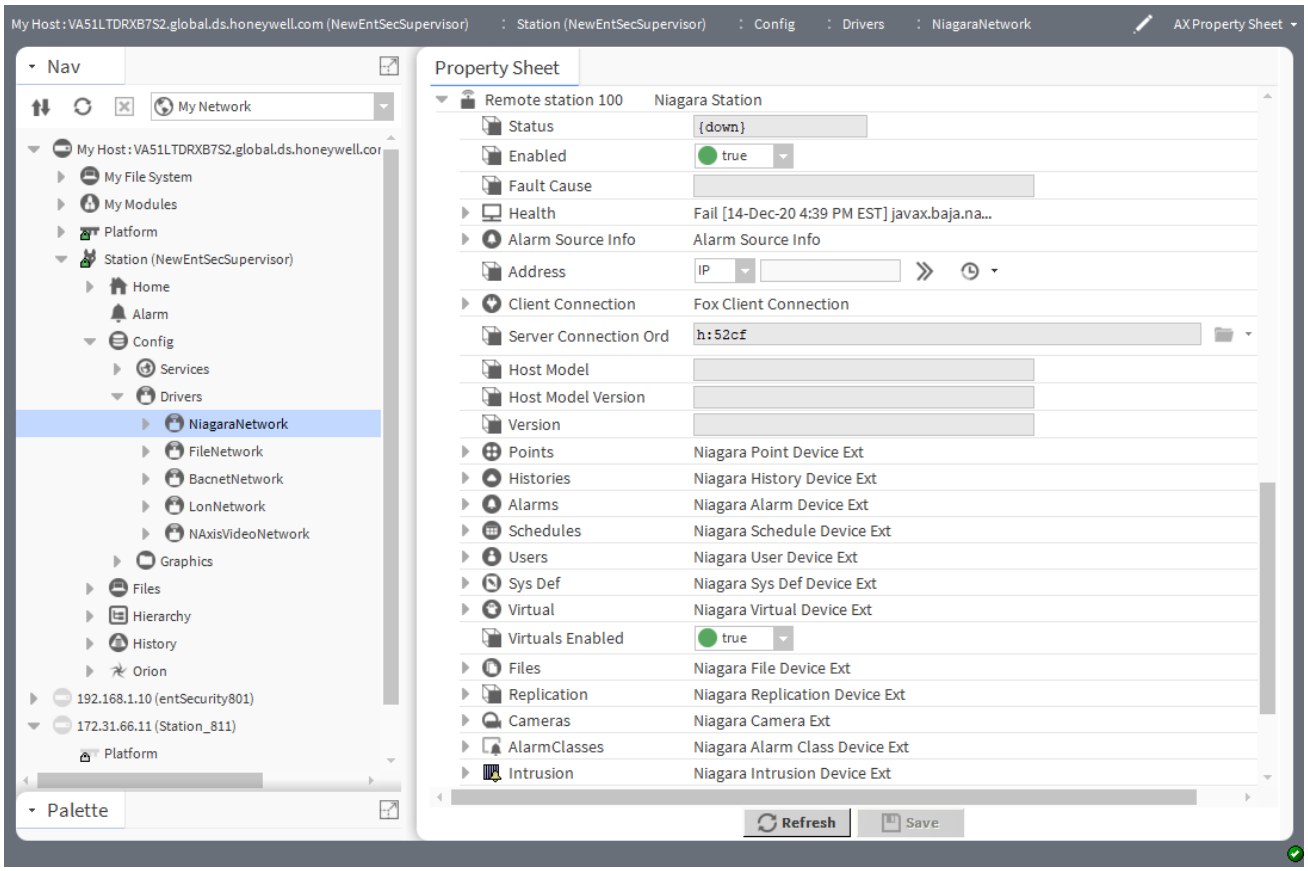
This component models other stations (**NiagaraStations**) under its **Drivers** container. These are the remote stations that a local station can communicate with. Its primary view is the **Station Manager**.

In general, a **NiagaraNetwork** uses the same driver architecture as other (non-NiagaraNetwork) drivers. By default it has a **NiagaraStation** component. A **NiagaraNetwork** differs from other networks in that:

- All proxy points under a **NiagaraStation** are read-only points: BooleanPoint, EnumPoint, NumericPoint, StringPoint. And a proxy point inherits any actions from the remote (source) point or object.
- Connections between stations occur as client and server sessions using the Fox protocol. The requesting station is the client, and target (responding) station is the server. Workbench is always a client and its station is the server. All Fox connections require client authentication (performed by the server).

  NOTE: The Fox protocol runs over a TLS-encrypted connection with certificate-based server authentication. This secure Foxs is noted and mentioned in various subsections of this document.

Figure 62 NiagaraNetwork properties



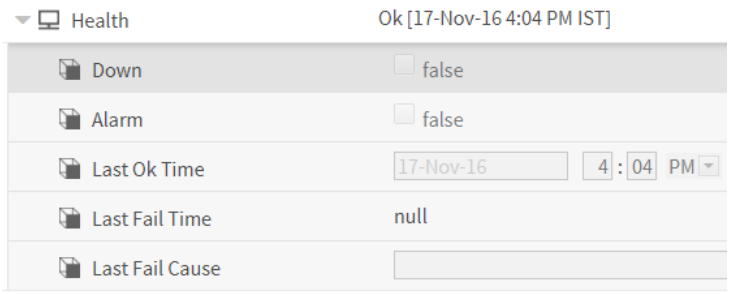
To view these properties, expand the **Config→Drivers**, right-click the **NiagaraNetwork** and click **Views→AX Property Sheet**.

Property	Value	Description
Status	read-only	Reports the condition of the entity or process at last polling. `{ok}` indicates that the component is licensed and polling successfully. `{down}` indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. `{disabled}` indicates that the **`Enable`** property is set to `false`. `{fault}` indicates another problem. Refer to **`Fault Cause`** for more information.
Enabled	`true` or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Fault Cause	read-only	Indicates the reason why a system object (network, device, component, extension, etc.) is not working (in fault). This property is empty unless a fault exists.
Health	additional properties	Reports the status of the network, device or component. This advisory information, including a time stamp, can help you recognize and troubleshoot problems but it provides no direct management controls. The *Niagara Drivers Guide* documents the these properties.
Alarm Source Info	additional properties	Contains a set of properties for configuring and routing alarms when this component is the alarm source. For property descriptions, refer to the *Niagara Alarms Guide*
Monitor	additional properties	Configures a network's ping mechanism, which verifies network health. This includes verifying the health of all connected objects (typically, devices) by pinging each device at a repeated interval. The *Niagara Drivers Guide* documents these properties. The **driver-PingMonitor** topic documents the monitor properties.
Local Station	additional properties	Serves as a container for read-only system definition (Sys Def) properties that reflect information synchronized with if the remote local station is defined as a subordinate. Sys Def properties are of use to developers. The **niagaraDriver-LocalSysDefStation** topic documents the additional properties.
SysDef Provider	additional properties	Serves as a container for system definition (Sys Def) child components of the type ProviderStations, which are hidden slots by default. The **niagaraDriver-BogProvider** topic documents the additional properties.
Tuning Policies	additional properties	Configures network rules for evaluating both write requests to writable proxy points as well as the acceptable freshness of read requests. For more information, refer to the *Niagara Drivers Guide*.

Property	Value	Description
		The **niagaraDriver-NiagaraTuningPolicyMap** topic documents properties.
History Policies	additional properties	Provides a container for the rules that specify how remotely-generated histories should be changed when exported into another station (usually the Supervisor station). The container also contains a poll scheduler for the on-demand polling of histories. The rules identify the source stations and their histories files to import from remote stations and define storage limits in the target station. The **driver-HistoryNetworkExt** topic documents history policy properties.
Workers	additional properties	Configures the maximum number of concurrent threads and maximum allowable queue size for the database connection. Tuning is for large **NiagaraNetwork**s and is the only visible part of a shared thread pool scheme, which provides large-job scalability. A single station may support more than one thread and the number of threads grows uncapped while alleviating a former thread starving issue. Very large Supervisors with many stations may benefit from thread pool adjustments made via entries in the host's `system.properties` file. Consult your support channel for details. Third–party drivers do not pool these worker properties at the network level as does the **NiagaraNetwork** driver. Each database driver has its own thread pool setting. In addition, tuning policies are simplified in a **NiagaraNetwork**, as compared to polling–type drivers. The **niagaraDriver-CyclicThreadPoolWorker** documents the **`Max Threads`** property.
Virtual Policies	additional properties	Configures how to handle Px files that include virtual objects as well as some other virtual settings. The **niagaraDriver-NiagaraVirtualNetworkExt** documents the additional properties.
Persist Fetched Tags	`true` or `false` (default)	Controls the persistence of proxy point tags. `true` causes the `n:history` tag (and any other tags configured for fetching) to persist on the proxy control point as read-only metadata (as a direct tag). `false` ignores proxy point tags.
tagsToFetch (optional)	comma delimited names	Identifies additional remote tags to fetch for the proxy points (and virtuals).
persistVirtual-FetchedTags (optional)	`true` or `false` (default)	Adds fetched tags under the level of this property's container as properties (read-only) for virtuals.

Health

These properties are common to all Niagara components. They report information about the current condition of the component.



Property	Value	Description
Down	`true` or `false` (default)	Displays the health of the network.
Alarm Source Info	additional properties	Contains a set of properties for configuring and routing alarms when this component is the alarm source. For property descriptions, refer to the *Niagara Alarms Guide* For more information, refer to the *Niagara Alarms Guide*
Last Fail Cause	text	Displays the reason for the last failure of the network health.
Last Fail Time	`date time`	Displays the last date and time the network health failed.
Last OK Time	`date time`	Displays the last date and time the network health was `{ok}`.

Actions

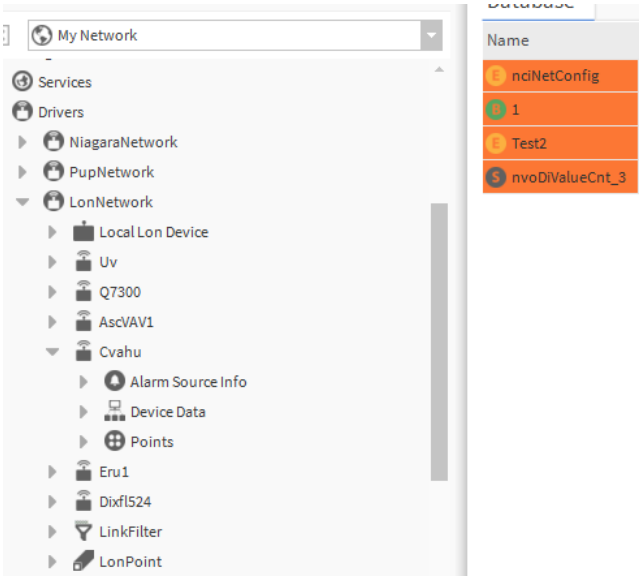
- **Ping** sends a message to a network object (device, database, etc). The message provokes a response, which indicates the current state of the object.
- **Reset All Connections** resets all connections within the network.
- **Submit Station Discovery Job** initiates a discovery job to populate the **Station Manager** view.
- **Force Update Niagara Proxy Points** manually accesses and returns proxy point data to the Supervisor station. In a remote station, this action returns data identified with an `n:history` tag. This action is available on the following components: **NiagaraNetwork**, **NiagaraStation**, **NiagaraStationFolder**, **NiagaraPointDeviceExt**, and **NiagaraPointFolder**.

niagaraDriver-NiagaraPointDeviceExt

This component is an implementation of PointDeviceExt. Its primary view is the **Point Manager**.

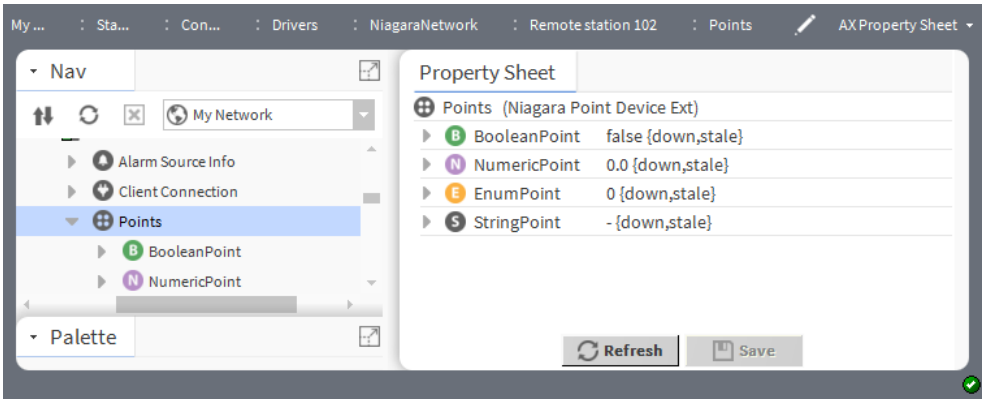
A device's Points extension (or Points) serves as the top parent container for real-time data values originating from the device.

Figure 63 Points under a Lon device



These values are proxied using framework control points, or proxy points. Values can be both read from data values in that device and written to value stores in the device.

Figure 64 Points properties



To open this **Property Sheet**, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Points** and click **Views→AX Property Sheet**.

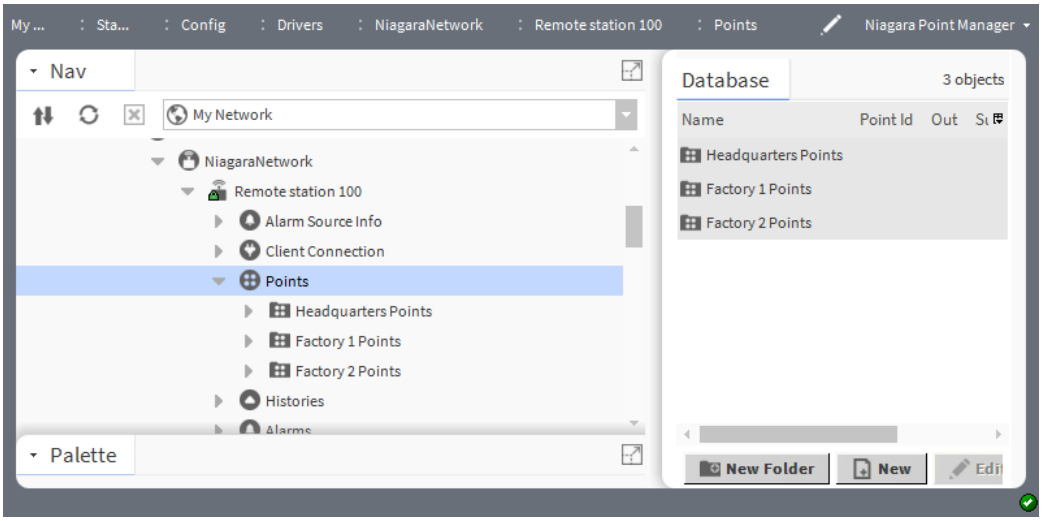
The **Points** folder contains no unique properties.

niagaraDriver-NiagaraPointFolder

This folder is an implementation of a folder under the **NiagaraStation**'s **Points** extension. You use it to group related points under the **Points** extension.

Each **NiagaraPointFolder** has its own **Point Manager** view. You add such folders using the **New Folder** button in the **Point Manager** view of the **Points** extension.

Figure 65 Points folder

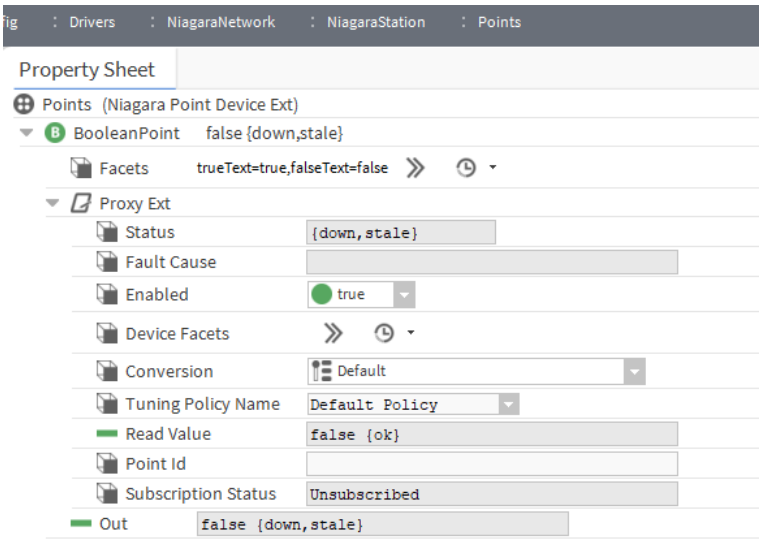


To add one of these folders, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, double-click **Points** and click **New Folder**, or drag a **PointFolder** form the **`niagaraDriver`** palette.

niagaraDriver-NiagaraProxyExt

This component is an implementation of BProxyExt and applies to controller stations. An alternative for Supervisorstations exists among the virtual components.

Figure 66 ProxyExt properties



To view these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Points**, click **Views→AX Property Sheet**, expand a point and double-click its **Proxy Ext**.

Property	Value	Description
Device Facets	additional properties	Determines additional parameters to config facets.
Conversion	drop-down list	Defines how the system converts proxy extension units to parent point units. `Default` automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point. **NOTE:** In most cases, the standard `Default` conversion is best. `Linear` applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the Scale and Offset properties to convert the output value to a unit other than that defined by device facets. `Linear With Unit` is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion. `Reverse Polarity` applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output. `500 Ohm Shunt` applies to voltage input points only. It reads a 4-to-20mA sensor, where the Ui input requires a 500 ohm resistor wired across (shunting) the input terminals. `Tabular Thermistor` applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors. `Thermistor Type 3` applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors. `Generic Tabular` applies to 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.
Tuning Policy Name	drop-down list	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.
Read Value	read-only	Displays the last value read from the device, expressed in device facets.
Point Id	read-only	Reports the point ID.

Property	Value	Description
Subscription Status	read-only	Reports the subscription status of the component.
Out	read-only	Displays the current value of the proxy point including facets and status. The value depends on the type of control point. Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level. Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as `fault`, `overridden`, `alarm`, and so on. If no status flag is set, status is considered normal and reports `{ok}`.

niagaraDriver-NiagaraScheduleDeviceExt

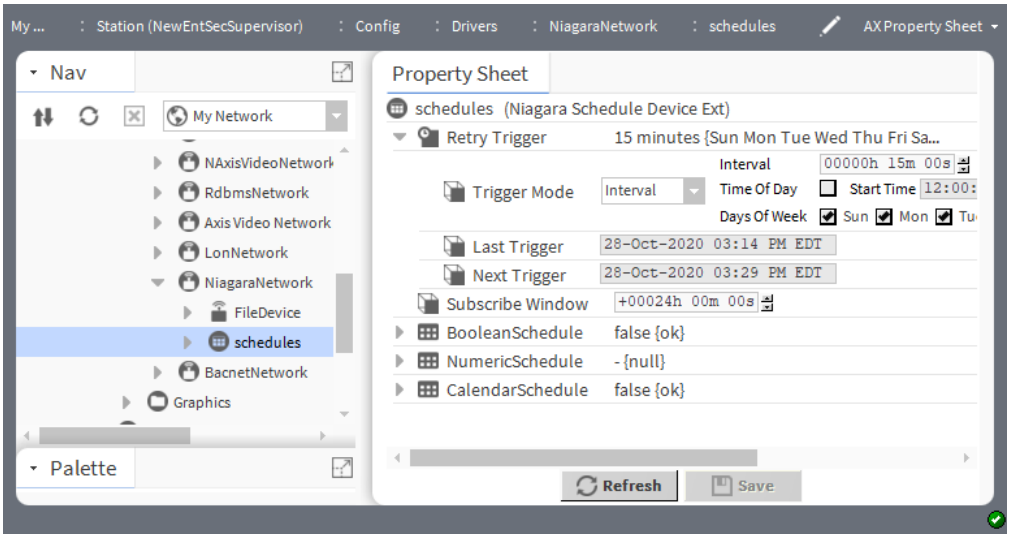
This component is the implementation of a schedules device extension. Not all drivers have devices with a **ScheduleDeviceExt** (Schedules). Notably, a **ScheduleDeviceExt** exists only if the driver protocol (or device) provides local event scheduling, which is the case with NiagaraStation and BacnetDevice components.

A device's schedules extension is the parent container for imported schedules—components with a ScheduleImportExt. The schedule imports events from the remote station. The imported schedule is read-only (often called a slave schedule). By default, the schedules extension contains a **Retry Trigger**.

A schedules extension is local to the station, can reside anywhere under the station's **Config** node and can be imported by one or more other stations. If you import a schedule from another **NiagaraStation** or BacnetDevice, it must reside in that device's **Schedules** container. Imported schedules are always under a specific device.

The schedules extension can also contain schedule export descriptors. These correspond to local station schedules (often called master schedules) that are exported (pushed) to a remote station. The framework automatically creates a schedule export descriptor when a local schedule is pushed to a remote station.

Figure 67 Schedule device extension properties



To view these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Schedules** and click **Views→AX Property Sheet**.

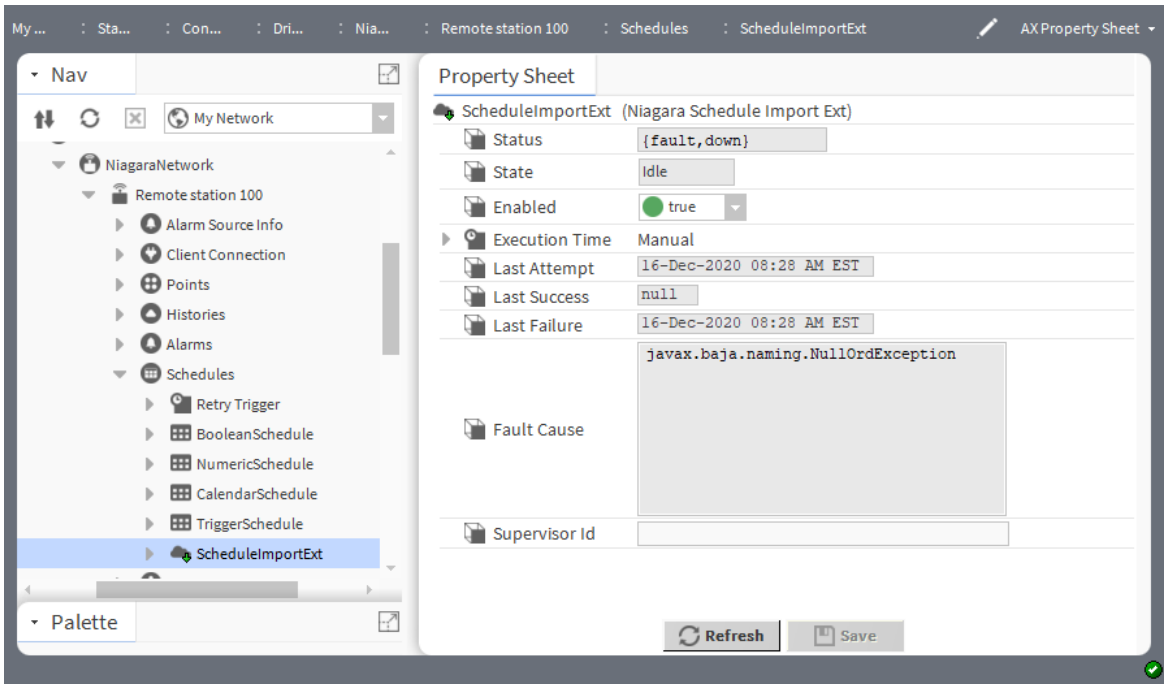
Property	Value	Description
Retry Trigger	additional properties	Defines how frequently to attempt a failed operation again. This continues until successful execution occurs. Appears in the Nav tree but not in any manager view and is unique in that it requires no linking of its output for operation. *Getting Started with Niagara* documents **`Retry Trigger`** properties.
Subscribe Window	hours minutes seconds (defaults to 24 hours)	Sets up a period of time during which subscription is possible.

niagaraDriver-NiagaraScheduleImportExt

This component defines in the local station the import properties for a remote schedule. You include this component in the station, usually a remote controller station, that receives a schedule, usually from a Supervisor station.

This component is in the **`niagaraDriver`** palette.

NiagaraScheduleImportExt properties



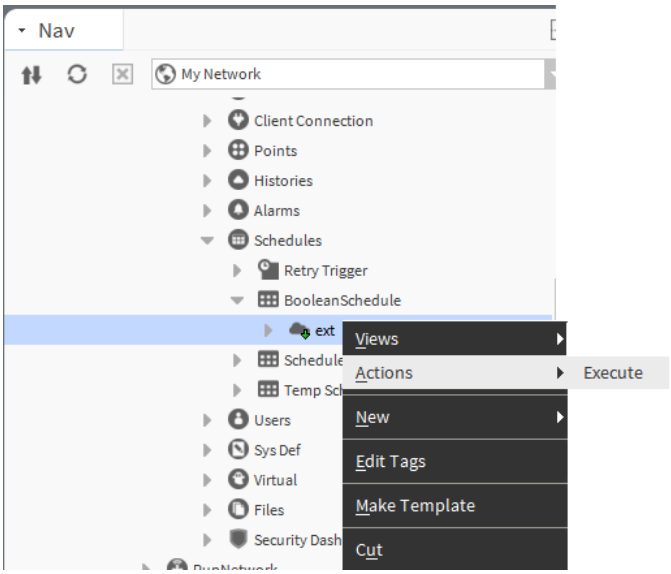
To view these properties, expand the **Config→Drivers→NiagaraNetwork**, expand a **NiagaraStation**, locate your **ScheculeImportExt** component and double-click it.

In addition to the standard properties (Status, Enabled and Fault Cause) these properties configure this component.

Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.
Supervisor Id	text	Provides information about this specific Supervisor station.

Action

Figure 68 actions menu



This component has one action, **Execute**. This action imports the selected schedule file(s).

niagaraDriver-NiagaraStation

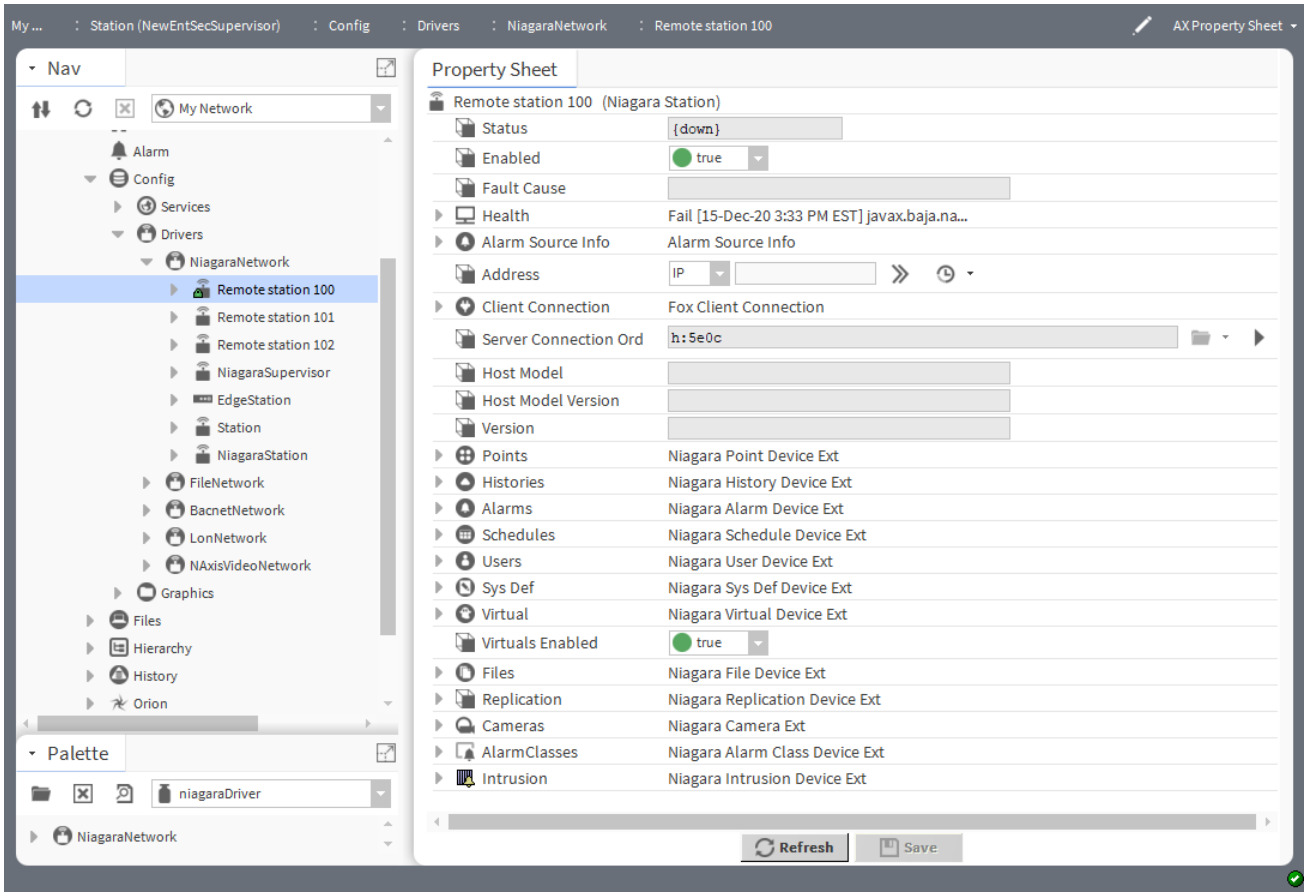
This component models a remote station in a Supervisor station.

While the connection may use Fox (not secure), a **NiagaraStation** with a padlock icon models a remote station via a secure TLS Foxs connection. This is the recommended connection.

Each **NiagaraStation** contains the standard set of device extensions. In addition it contains:

- **Users** device extension. For more information, refer to the **niagaraDriver-NiagaraUserDeviceExt** topic.
- **Virtuals** (gateway) slot, and an associated **`Virtuals Enabled`** property.

Figure 69 NiagaraStation properties



To view these properties, expand **Config→Drivers→NiagaraNetwork** and double-click a station.

In addition to the standard properties (Status, Enabled, Fault Cause, Health and Alarm Source Info), these properties are unique to this component. A separate topic documents each extension.

Property	Value	Description
Address, type	drop-down list (defaults to `IP`)	Selects the type of address.
Address, value	IP address	Identifies the station's IP address.
Client Connection	additional properties	Configure a FoxSession connection. The "fox-FoxClientConnection" topic documents these properties.
Server Connection Ord	read-only	Reports a connection.
Host Model	read-only	Reports the host model.
Host Model Version	read-only	Reports the version number of the host model.
Version	read-only	Reports a version number.
Virtuals Enabled	`true` (default) or `false`	Turns support for virtuals on (`true`) and off (`false`)

Actions

Ping —Sends a message to the device.

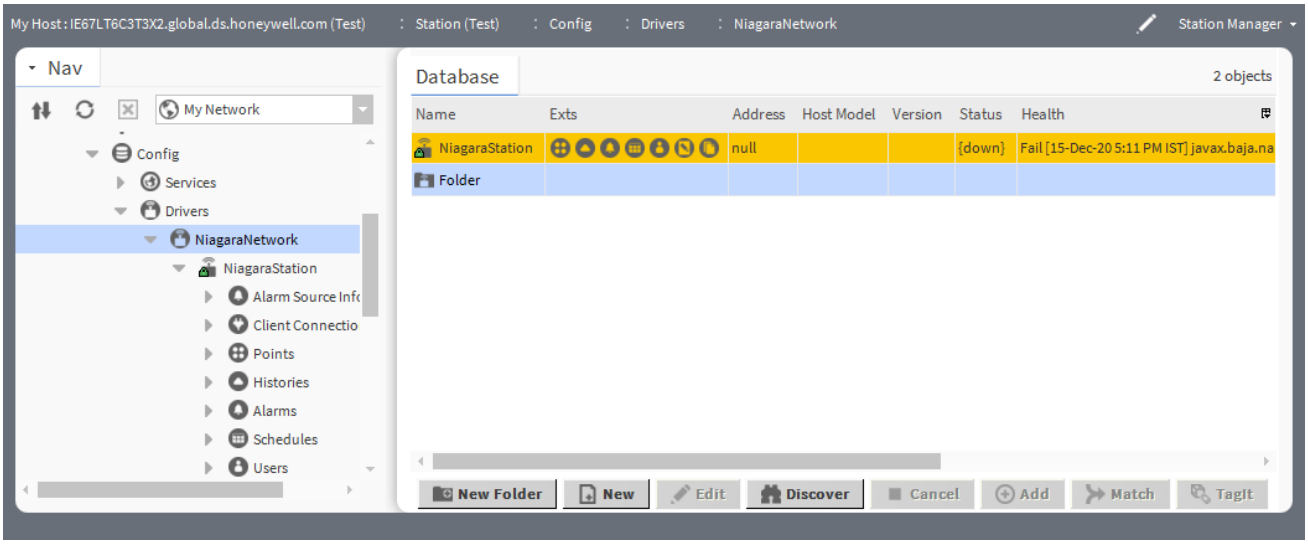
Force Update Niagara Proxy Points — In Niagara 4.2 and later, forces a re-fetch of `n:history` tag information. The action is available on the following components: **NiagaraNetwork**, **NiagaraStation**, **NiagaraStationFolder**, **NiagaraPointDeviceExt**, and **NiagaraPointFolder**.

niagaraDriver-NiagaraStationFolder

This component is an implementation of a folder under the **NiagaraNetwork**. You use it to group stations under the **NiagaraNetwork**.

Each**NiagaraStationFolder** has its own **Station Manager** view. Station folders can be useful in very large systems to organize **NiagaraStation** components.

Figure 70 NiagaraStationFolder



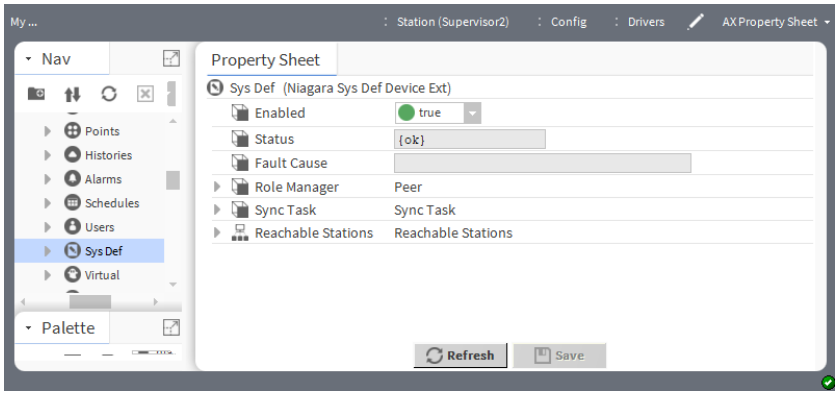
To add one of these folders, expand **Config→Drivers**, double-click **NiagaraNetwork** and click **New Folder** in the **Station Manager**, or drag a **PointFolder** form the **`niagaraDriver`** palette..

To select this view right-click **NiagaraNetwork** and click **ViewsStation Manager**.

niagaraDriver-NiagaraSysDefDeviceExt

This component (**Sys Def**), which the station includes among its device extensions under any **NiagaraStation**, is chiefly of interest to developers. Together with the **NiagaraNetwork**'s Sys Def Provider (BogProvider) component, the API helps arrange stations in the **NiagaraNetwork** using a known hierarchy, which allows for the synchronization of basic information up the hierarchy.

Figure 71 Niagara Station Sys Def properties



To access this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click the **Sys Def** component in the Nav tree, and click **Views→AX Property Sheet**.

In addition to the standard properties (Enabled, Status and Fault Cause), these properties are unique to this component.

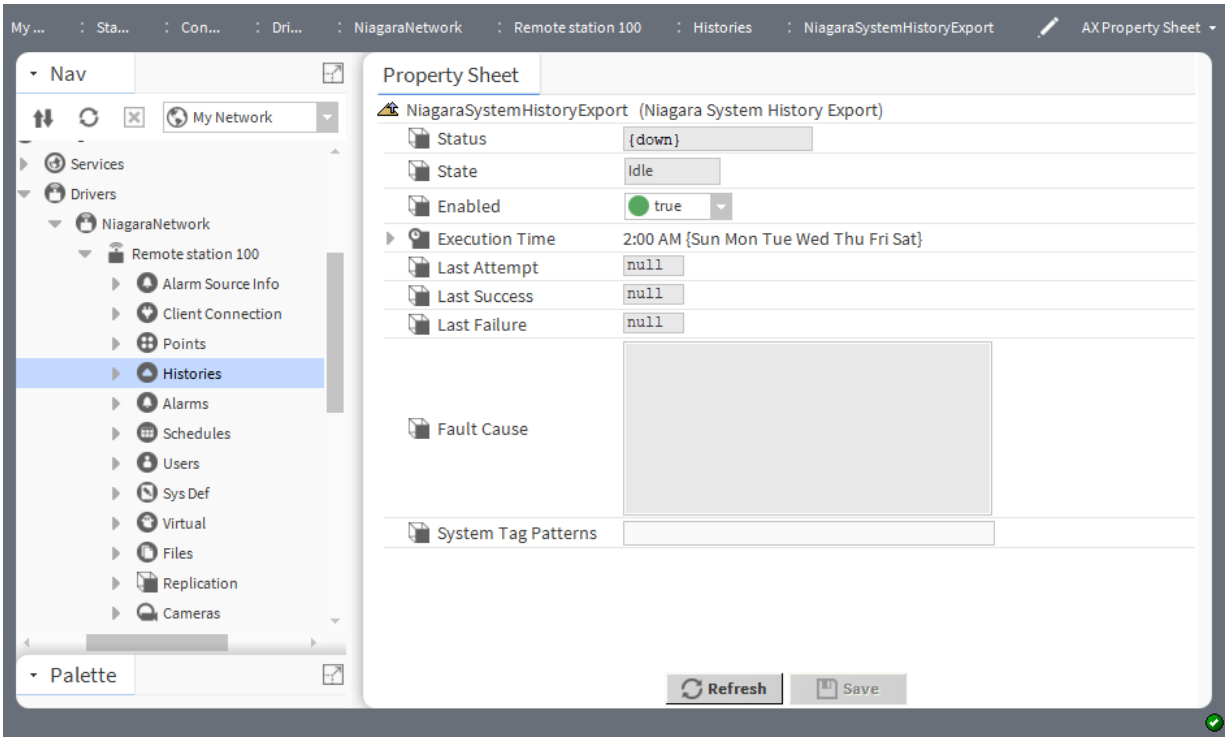
Property	Value	Description
Role Manager	additional properties	Configures roles. The niagaraDriver-RoleManager topic in this guide documents these properties.
Sync Task	additional properties	Configures the synchronization task. The niagaraDriver-SyncTask in this guide documents these properties.
Reachable Stations	additional properties	As of Niagara 4.13, defines all of the downstream NiagaraStations that are routable starting at the NiagaraStation in which this container resides. To learn more about this container component, see “Viewing reachable stations” in “Niagara System Database and System Indexing Guide”.

niagaraDriver-NiagaraSystemHistoryExport

This component defines the system tags text patterns used to export local histories into the target **NiagaraStation**. **NiagaraSystemHistoryExport** descriptors reside under the **Histories** extension of the **NiagaraStation** in the **NiagaraNetwork**.

System history export descriptors configure the system tags properties of local history extensions instead of unique history IDs.

Figure 72 NiagaraSystemHistoryExport properties



Before you can view these properties you must have already created a system history export descriptor. To view these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click Histories and click **Views→AX Property Sheet** and double-click the system history export row in the table.

In addition to the standard properties (Status, Enabled and Fault Cause), these properties support this component.

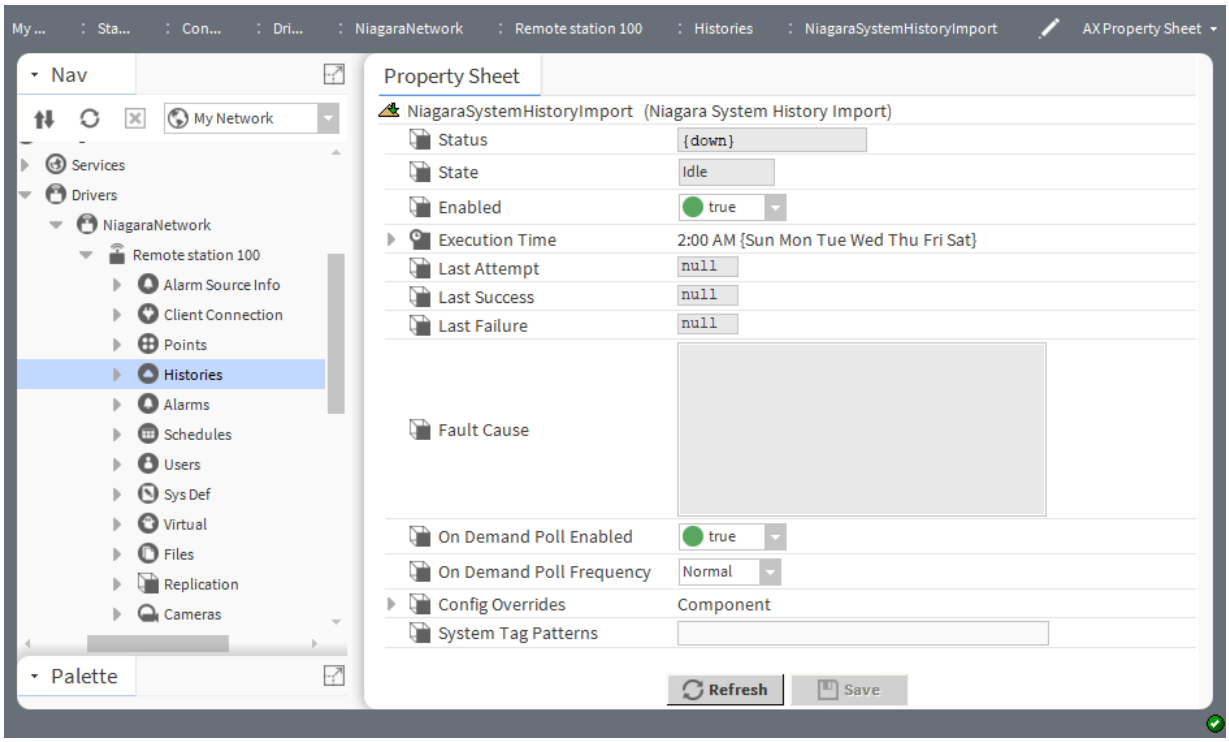
Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.
System Tag Patterns	text	Specifies one or more text strings matched against text values in the System Tags properties of a local history extensions, where matching text patterns result in histories exported into the remote history space.

niagaraDriver-NiagaraSystemHistoryImport

This component defines the system tags text patterns used to import remote histories from the source **NiagaraStations** into the target **NiagaraStation**. **NiagaraSystemHistoryImport** descriptors reside under the **Histories** extension of the **NiagaraStation** in the **NiagaraNetwork**.

System history export descriptors configure the system tags properties of the remote history extensions instead of unique history IDs.

Figure 73 NiagaraSystemHistoryExporImport properties



Before you can view these properties you must have already created a system history import descriptor. To view these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation→Histories** and double-click the system history import row in the Nav tree.

In addition to the standard properties (Status, Enabled and Fault Cause), these properties support this component.

Property	Value	Description
State	read-only	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Execution Time	additional properties	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Last Success	read-only	Reports the date and time of the last successful execution.
Last Failure	read-only	Reports the date and time of the last execution failure.

Property	Value	Description
On Demand Poll Enabled		Enables and disables polling. `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
On Demand Poll Frequency		References the **`On Demand Poll Scheduler`** rates under the **NiagaraNetwork**'s **History Policies** container slot.
Config Overrides, capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Config Overrides, fullPolicy	drop-down list (defaults to `Roll`)	Defines what happens if **`Capacity`** is set to **`Record Count`** and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.
System Tag Patterns		Specifies one or more text strings matched against text values in the System Tags properties of a local history extensions, where matching text patterns result in histories exported into the remote history space.

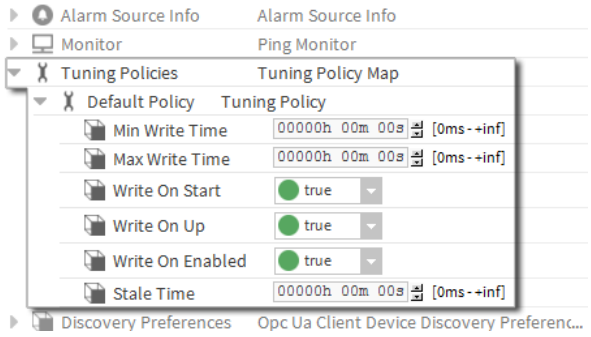
niagaraDriver-NiagaraTuningPolicy

Contains properties used in the NiagaraNetwork's handling of both write requests (e.g. to writable proxy points) as well as the acceptable freshness of read requests of proxy points. You can create multiple tuning policies under the NiagaraNetwork's NiagaraTuningPolicyMap. You can then assign one or more proxy points to a specific policy. Container for one or more NiagaraTuningPolicy(ies), found in the NiagaraNetwork's property sheet.

A network's **`Tuning Policies`** holds one or more collections of rules for evaluating both write requests (e.g. to writable proxy points) as well as the acceptable freshness of read requests from polling. In some drivers (such as Bacnet), also supported is association to different poll frequency groups (Slow, Normal, Fast). Tuning policies are important because they can affect the status of the driver's proxy points.

In the network's property sheet, expand the **`Tuning Policies (Map)`** slot to see one or more contained Tuning Policies. Expand a Tuning Policy to see its configuration properties, as shown.

Figure 74 Example Tuning Policies Map (OpcUa)



NOTE: Some driver networks do not have Tuning Policies, for example an RdbmsNetwork for a database driver.

By default, a driver's TuningPoliciesMap contains just a single TuningPolicy (Default Policy). However, you create multiple tuning policies, changing those items needed differently in each one. Then, when you create proxy points under a device in that network, you can assign each point (as needed) to the proper set of "rules" by associating it with a specific tuning policy.

CAUTION:

Using only a single (default) tuning policy, particularly with all property values at defaults, can lead to possible issues in many driver (network) scenarios. In general, it is recommended that you create multiple tuning policies, and configure and use them differently, according to the needs of the network's proxy points and the capabilities of the driver. In particular, tuning policy properties that specify writes from the framework should be understood and applied appropriately. See Tuning Policy properties for more details.

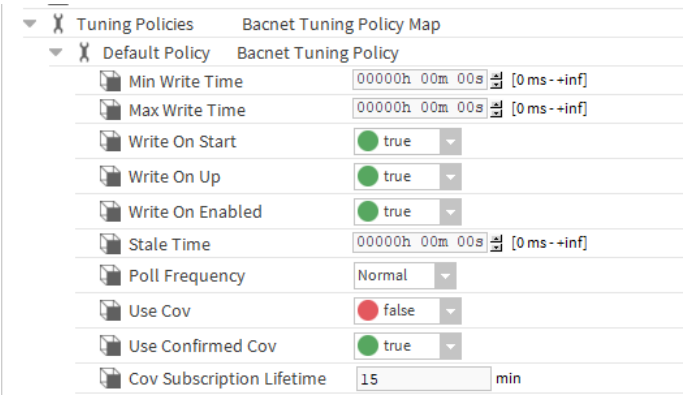
As a simple example (under a BacnetNetwork), you could change the default tuning policy's **`Write On Start`**property from the default (`true`) to `false`. Then, duplicate the default tuning policy three times, naming the first copy Slow Policy, the second copy Normal with Write Startup, and the third copy Fast Policy. In two of those copies, change the **`Poll Frequency`**property from Normal to Slow or Fast, corresponding to its name. In the Normal with Write Startup tuning policy copy, you could change its **`Write On Start`** property back to `true`.

Then, only the Normal with Write Startup tuning policy has Write On Start set as true. At this point you would then have four available (and different) tuning policies to pick from when you create and edit proxy points, where you could selectively apply the policy needed.

Tuning Policy Properties

Tuning policies used by a specific driver may have unique characteristics. Other drivers may have specific considerations for tuning policies. The below is the example of Bacnet driver tuning policy properties:

Figure 75 Example of tuning policy properties for the Bacnet driver



You access this **Property Sheet** by expanding **Config→Drivers** followed by double-clicking or right-clicking the network node, clicking **Views→Property Sheet** and expanding the **Tuning Policies→Default Policy**.

Property	Value	Description
Min Write Time	hours minutes seconds (defaults to zero (0))	Specifies the minimum amount of time allowed between writes to writable proxy points, especially ones that have one or more linked inputs. This provides a way to throttle rapidly changing values so that only the last value is written. The default value (0) disables this rule causing all value changes to attempt to write.
Max Write Time	hours minutes seconds (defaults to zero (0))	Specifies the maximum amount of time to wait before rewriting the value, in case nothing else has triggered a write, to writable proxy points. Any write action resets this timer. The default (0) disables this rule resulting in no timed rewrites. In some cases, setting this to a value, such as 10 minutes, may be useful. Often, a network may have devices that, upon a power cycle or even a power interruption, have writable points that reset to some preset default value or state. Often, in a site-wide power bump of a few seconds, such field controllers (devices on the network) typically reset, but the controller continues normal operation running on a backup battery. Since the network's default monitor ping is usually 5 minutes, the station (network) may never mark these devices as down, such that a **`Write On Up`** does not occur. If a writable point represents an AHU or chiller that defaults to unoccupied following a device reset, the load never restarts until the next day when the schedule toggles. Assigning the point to tuning policy that does have a configured **`Max Write Time`** can correct this problem. At the same time, realize that many networks may be configured such that multiple masters may be issuing conflicting writes to one or more points in a device. In this case, exercise caution with this property to avoid write contention that could result in toggling loads.
Write On Start	`true` (default) or `false`	Determines a writable proxy point's behavior when the station starts. `true` initiates a write when the station first reaches a steady state. `false` prevents a write when the station first reaches a steady state. **NOTE:** Consider setting to `false` except for critical proxy points, otherwise large networks may experience write-queue-overflow exceptions. Consider setting this to `false` for most tuning policies, except for tuning policies selectively assigned to more critical writable proxy points. This is particularly important for large networks with many writable proxy points. For example, a BacnetNetwork with 4,000 writable proxy points, if configured with only the default Tuning Policy (at default values), upon station startup attempts to write to all 4,000 points. This puts a significant load on the station. As a

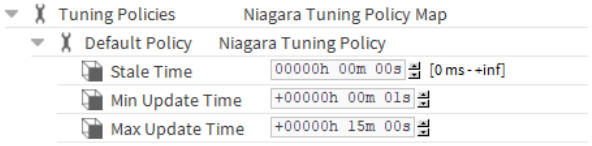
Property	Value	Description
		consequence, the BACnet driver (network) may generate write-queue-overflow exceptions.
Write On Up	`true` (default) or `false`	Determines a writable proxy point's behavior when the point and its parent device transition from down to up. `true` initiates a write when a transition from down to up occurs. `false` prevents a write when a transition from down to up occurs.
Write On Enabled	`true` (default) or `false`	Determines a writable proxy point's behavior when the point's status transitions from disabled to normal (enabled). `true` initiates a write when the transition occurs. `false` prevents a write when the transition occurs. The disabled-to-enabled status transition can be inherited globally by points if the parent device was set to disabled, or network-wide, if the driver network was set to disabled. Therefore, be aware that when left at `true` all associated writable proxy points receive a write when either the device or network transition from a status of disabled to enabled.
Stale Time	hours minutes seconds (defaults to zero (0))	Defines the period of time without a successful read (indicated by a read status of {ok}) after which a point's value is considered to be too old to be meaningful (stale). A non-zero value causes the point to become stale (status stale) if the configured time elapses without a successful read, indicated by Read Status `{ok}`. The default value (zero) disables the stale timer causing points to become stale immediately when unsubscribed. A tan background identifies stale proxy points. A stale status is considered invalid for any downstream-linked control logic. Do not configure an amount of time shorter than the poll cycle time. If you do, points will go stale in the course of normal polling. Instead, set this time to be longer than the largest expected poll cycle time. You should configure a Stale Time that is at least three times the expected poll cycle time. Most peer-to-peer networks do experience collisions and missed messages. Setting the Stale Time short will likely produce nuisance stale statuses. If a message is missed, a longer Stale Time allows for another poll cycle time during which to receive the message before setting the stale flag.
Poll Frequency (This property may not exist in some driver's tuning policies, but is instead a separate property of each ProxyExt)	drop-down list (defaults to `Normal`)	Selects among three rates (Fast, Normal and Slow) to determine how often to query the component for its value. The network's Poll Service or Poll Scheduler defines these rates in hours, minutes and seconds. For example: `Fast` may set polling frequency to every second. `Normal` may set poll frequency to every five seconds. `Slow` may set poll frequency to every 30 seconds.

Property	Value	Description
		This property applies to all proxy points. Depending on the driver, there may be a single Poll Service (or Poll Scheduler) slot under the network, or as in the case of a BacnetNetwork, a separate Poll Service for each configured port (IP, Ethernet, Mstp) under its **BacnetComm→Network** container. The **NiagaraNetwork** uses subscriptions instead of polling.
Use Cov	`true` or `false` If the device was discovered, and station database determined that the device indicates support for server-side COV, this property defaults to `true`. Otherwise, it defaults to `false` indicating that no proxy points under the device use COV.	Enables and disables a device's support for COV (change of value) as a way to monitor proxy point values. Assuming the device supports subscription to the COV service, `true` triggers the driver to attempt the necessary updates (proxy subscriptions) to the value of each point using the BACnet COV subscription to the device. If the subscription attempt succeeds, the **`Read Status`** property of the point's BacnetProxyExt displays `COV`. If the subscription attempt fails, the driver uses normal polling and the **`Read Status`** property shows `Polled`. When `true`, individual proxy points under the device may use COV subscriptions, depending on their assigned tuning policy. When `false`, the driver ignores any proxy subscription updates. How to find: expand the BACnet network and double-click the BACnet device. Also available on the Device Manager **Add** window.
Use Confirmed Cov	`true` (default) or `false`	Controls device updates. If enabled (`true`), and the assigned proxy points are under a BacnetDevice that supports Confirmed COV notifications, the driver attempts any necessary updates (proxy subscriptions) using BACnet confirmed COV subscriptions to the device. If disabled (`false`), the driver ignores this property. How to find: expand **Config→Drivers**, BACnet network, expand **Tuning Policies** and double-click **Default Policy**
Cov Subscription Lifetime	time range (defaults to 15 minutes)	Indicates the lifetime, in minutes, for which the database subscribes for COV notifications, then (if necessary) periodically subscribes again. A value of zero means an indefinite lifetime, although this is not guaranteed to persist across resets of the server device.

NiagaraNetwork Tuning Policies

A NiagaraNetwork has only three tuning policies as shown below:

Figure 76 NiagaraNetwork Tuning Policy

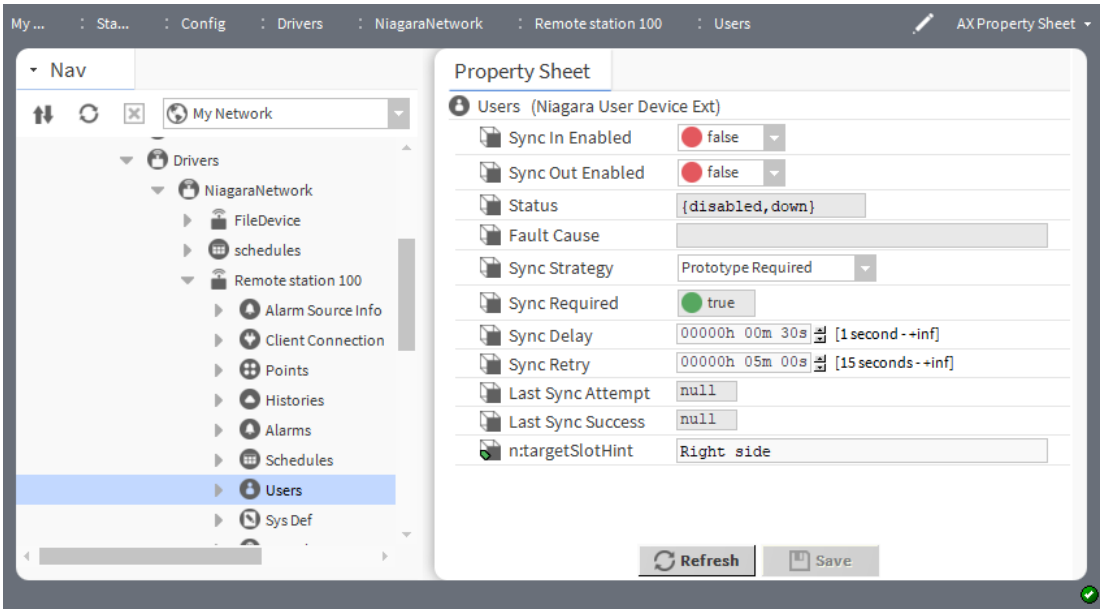


You access this Property Sheet by expanding **Config→Drivers** followed by double-clicking or right-clicking the network node, clicking **Views→Property Sheet** and expanding the **Tuning Policies→Default Policy**.

Property	Value	Description
Stale Time	time	If set to a non-zero value, a subscribed proxy point becomes stale (status stale) if the configured Max Update Time expires without an update from the server (source station). This stale timer is reset upon each subscription update. If set to zero (default), the stale timer is disabled, and a subscribed point becomes stale only while the source (server) point is also stale. **NOTE:** Whenever a source point of a proxy point has a stale status, for example a Bacnet proxy point, the proxy point for it will also have a stale status, regardless of this setting. Stale time is client side, whereas the other two update time properties affect server side operation of the subscription. For example, when a client (Supervisor) station creates subscriptions to a server station (JACE with a field bus), say to update proxy point values on a Px page, subscriptions are set up with the server to observe rules in the Min and Max Update Time values.
Min Update Time	Time	The minimum amount of time between updates sent from the server to the client. It is used to throttle data changing at a rate faster than minUpdateTime. Default value is 1 second.
Max Update Time	Time	Used by the server to resend the values to the client for subscribed points, if values have not been sent for other reasons (such as a change of value or status). Default value is 15 minutes. **NOTE:** Relative to tuning policies in other networks, the importance of **NiagaraNetwork** tuning policies are secondary, and then only applicable for a station that has proxy points under its **NiagaraNetwork**. In applications, this means the Supervisor station only. As a general rule, if configuring the Stale Time in a Tuning Policy, it is recommended to be greater than the Max Update Time by a factor of three.

niagaraDriver-NiagaraUserDeviceExt

This component manages user data for multi-station jobs. It enables and configures user synchronization both in and out of a station in relationship to a station's **NiagaraNetwork**. There is no special view (apart from the **Property Sheet**) for this extension, nor is it a container for other components. Instead, its properties configure the synchronization process for network users in the proxied station, relative to the current (remote) station.



To access these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation** and double-click **Users**.

In addition to the standard properties (Status and Fault Cause), these properties are unique to this component.

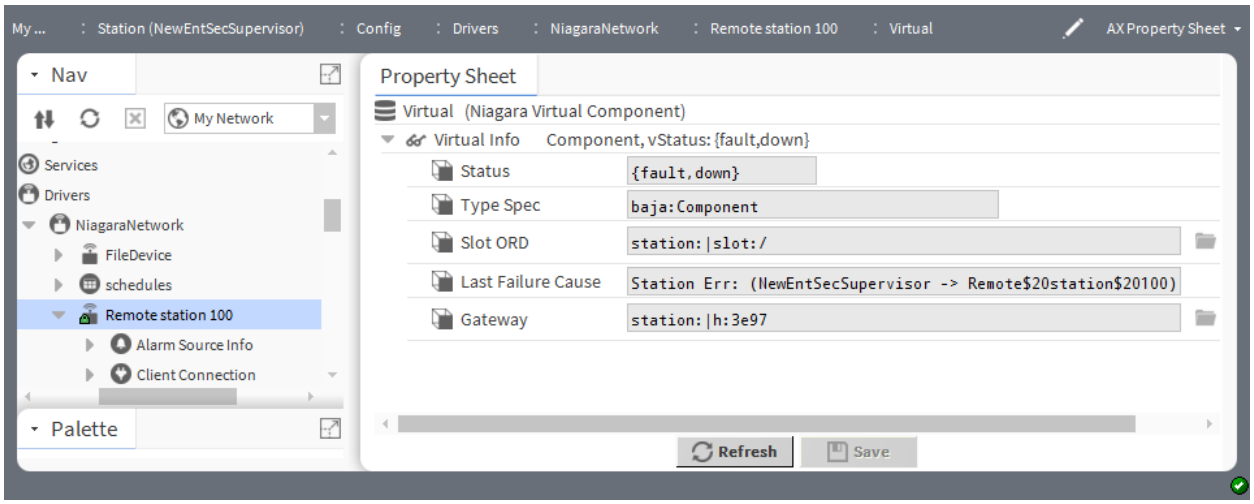
Property	Value	Description
Sync In Enabled	`true`, or `false` (default)	Enables (`true`) and disables (`false`) the sending of user data from the proxied station (usually the Supervisor station) to the current (receiving) station (usually a remote controller station). **NOTE:** Niagara 4 supports only sync-in between N4 stations.
Sync Out Enabled	`true` or `false` (default)	Enables (`true`) and disables (`false`) the receiving of user data from a current station (usually the remote controller station) to the proxied station (usually a **NiagaraStation** under the Supervisor's **NiagaraNetwork**). **NOTE:** Niagara 4 supports sync-out among multiple Niagara 4 stations and from N4 stations to AX stations.
Sync Strategy	drop-down list (defaults to `Prototype Required`)	Controls how sync-in and sync-out work. `Prototype Required` does not add or modify the user on this proxied station unless the current (remote) station has a matching (identically-named) user prototype as referenced in the source user's **`Prototype Name`** property. **`Use Default Prototype`** always adds or modifies a network user. If the remote station has a user prototype named the same as the user's **`Prototype Name`** it uses the user prototype, otherwise it uses the **`Default Prototype`** in the receiving station.
Sync Required	read-only	Indicates if pending user changes require synchronization to become effective. Ordinarily this value is `false` unless user changes have occurred and the **`Sync Delay`** time has not expired.

Property	Value	Description
Sync Delay	00000h 00m 30s (default)	Applies to a sending user station only, where it specifies an amount of time that counts down following a change to a network user. This value resets to full delay time upon each successive change. Following the last user change, if the delay time is met, the station sends the network user changes to the proxied station to synchronize the records. You can set a unique value for each proxied stations.
Sync Retry	00000h 05m 00s (default)	Applies to a sending user station only, where it specifies an amount of time that repeat time for previously unsuccessful user synchronization attempts (sent to the proxied station). Periodically, user synchronization attempts at this frequency until it successfully synchronizes.
Last Sync Attempt	read-only	Reports a timestamp that indicates when user synchronization to and from this station was last attempted.
Last Sync Success	read-only	Reports a timestamp that indicates when user synchronization to and from this station was evaluated as successful.

niagaraDriver-NiagaraVirtualDeviceExt

This component is the driver implementation of the Baja virtual gateway in a station. A virtual gateway is a component that resides under the station's component space (**Config**), and acts as a gateway to the station's virtual component space. Other object spaces are **Files** and **History**.

Figure 77 Virtual gateway properties



To access these properties, expand **Config→Drivers→NiagaraNetwork**, double-click a station component, set **`Virtuals Enabled`** to `true` and click **Virtual**.

In addition to the standard property, Status, these properties configure the virtual gateway.

Property	Value	Description
Type Spec	read-only	Reports the type of component.
Slot ORD	ORD	Configures the location of the slot in the station.

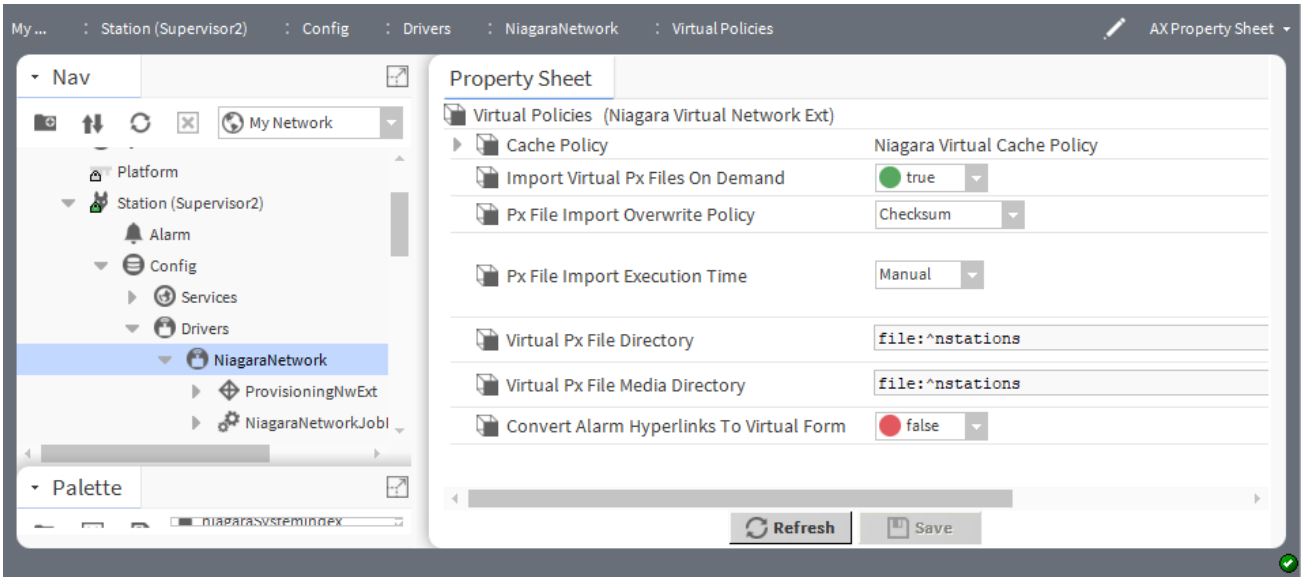
Property	Value	Description
Last Failure Cause	read-only	Reports the reason the virtual component failed most recently.
Gateway	ORD	Configures the location of the Gateway.

niagaraDriver-NiagaraVirtualNetworkExt

This component configures a **NiagaraNetwork**'s cache policies for virtual components.

For Supervisor stations that uses virtuals, this component configures cache properties and Px file policies. Remote stations do not require this component.

Figure 78 Niagara Virtual Network Ext properties



To access these properties, expand **Config→Drivers**, right-click the **NiagaraNetwork**, click **Views→AX Property Sheet**, click **Virtual Policies**.

Name	Value	Description
Cache Policy	additional properties	Configures how memory supports virtual components. Additional topics document these properties: "niagaraVirtual-NiagaraVirtualCachePolicy" and niagaraVirt"ual-DefaultNiagaraVirtualCache."
Import Virtual Px Files On Demand	`true`, `false` (default)	Enables (`true`) and disables (`false`) the importation of Px views on demand.
Px File Import Overwrite Policy	Drop-down list (defaults to `Checksum`)	Configures when to overwrite the file import descriptors that the driver created in the station automatically on demand. Px views on virtuals work by creating file import descriptors under the **NiagaraStation**'s **Files** device ext. `Checksum` `Last Modified` overwrites the descriptors that were modified last.
Px File Import Execution Time	Drop-down list (defaults to `Manual`)	Specifies the execution time interval to use on file import descriptors that the station creates automatically on demand.

Name	Value	Description
		`Manual` indicates that a user action triggers the import. `Daily` imports file descriptors at the defined time every day. **`Interval`** imports file descriptors when the specified amount of time passes.
Virtual Px File Directory	file:^nstations	Defines the root directory to which to import file import Px files when they are created on demand. The driver organizes imported Px files under this root directory by station name and file path as it existed on the remote station.
Virtual Px File Media Directory	file:^nstations	Sets up the root directory to which the driver imports Px media files (image files, etc) when it creates a file import descriptor. The driver organizes imported Px files under this root directory by station name and file path as it existed on the remote station.
Convert Alarm Hyperlinks To Virtual Form	`true`, `false` (default)	As of Niagara 4.13, if set to `true`, generated alarms routed up to the local station from another downstream station will have their hyperlink Ords converted to "nspace" form (when applicable). This conversion allows resolving alarm hyperlinks to Niagara virtuals on the local station instead of making a direct hyperlink connection to a subordinate station.

niagaraDriver-ProviderStation

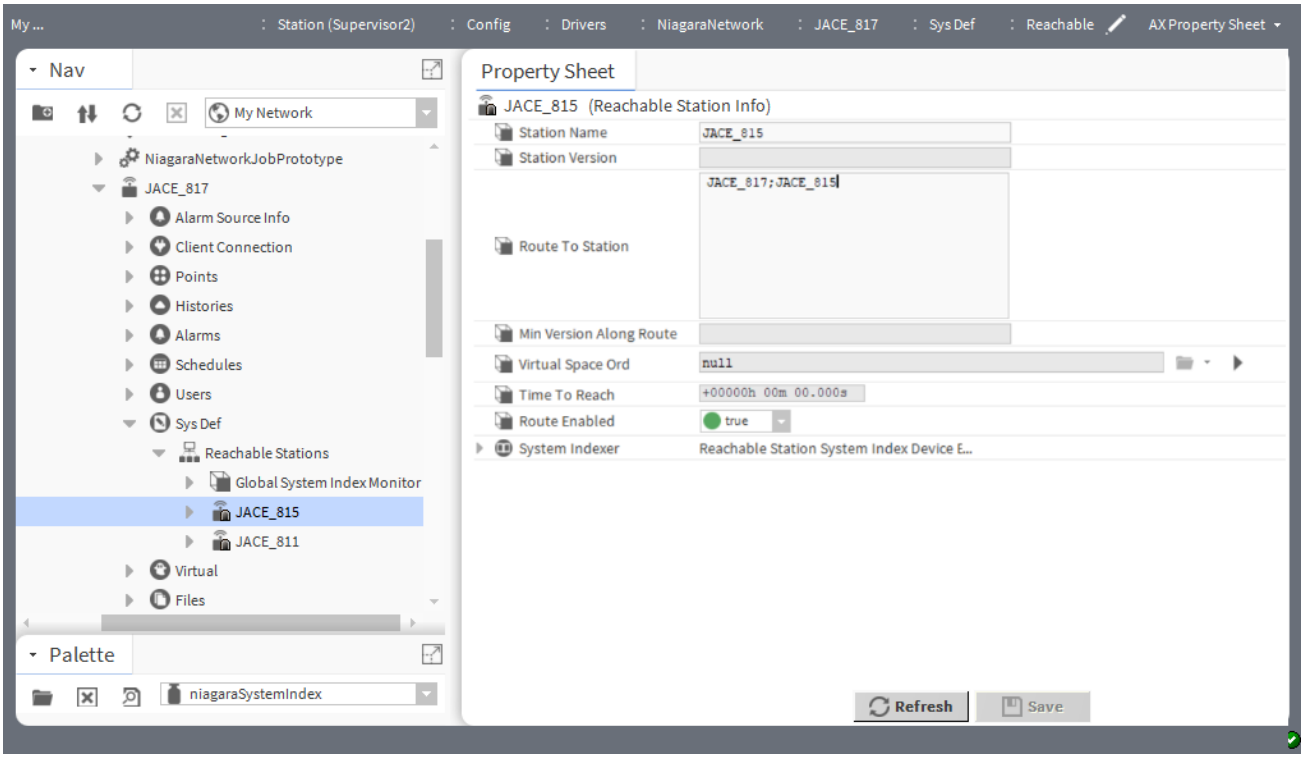
ProviderStation (stationName) is a child container under the **BogProvider** (Sys Def Provider) container in the NiagaraStation. Each reflects common “Sys Def” properties for the named station, which have been sync’ed up from remote subordinate stations (note that one **ProviderStation** mirrors the LocalSysDefStation). Note these components may be hidden, by default, as well as some of their child properties. Sys Def is chiefly of interest to developers extending the API.

niagaraDriver-ReachableStationInfo

This component appears automatically for any discovered, reachable station. It provides route information and allows you to enable or disable a particular route to a reachable station.

The ReachableStationInfo components get automatically placed under **“ Station”→Sys Def→ReachableStations**.

Figure 79 Reachable Station Info properties

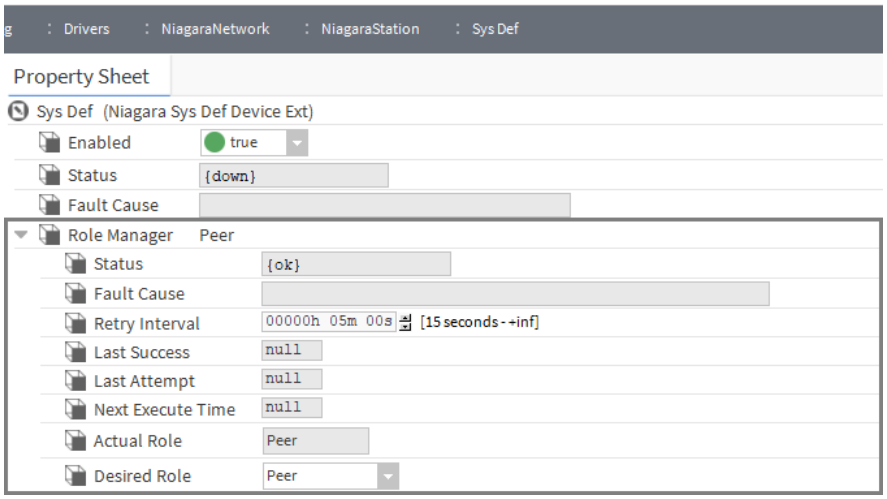


Property	Value	Description
Station Name	read-only	Displays the name of a remote, reachable station.
Station Version	read-only	Specifies the baja version of the destination reachable station.
Route To Station	read-only	The route of intermediate station names (delimited by semi-colons) to reach the remote, reachable station.
Min Version Along Route	read-only	Specifies the minimum baja version along the route of mid-tier stations to reach the destination reachable station.
Virtual Space Ord	read-only	Displays the ORD to the virtual space of the remote, reachable station. If virtuals are not enabled somewhere along the route, this ORD will be NULL.
Time To Reach	read-only	Displays the last amount of time it took to reach the remote, reachable NiagaraStation over the route during the last update.
Route Enabled	`true` (default), `false`	If set to `true`, this particular route to the reachable station is enabled and may be used, for example, for system indexing. If set to `false`, this route is disabled and will be excluded from the set of possible routes to a reachable station during activities such as system indexing.

niagaraDriver-RoleManager

RoleManager is child container under NiagaraStation's Sys Def device extension (NiagaraSysDefDeviceExt) that specifies and synchronize the relationship of the remote station to the local station. The "Sys Def" feature is chiefly of interest to developers extending the API.

Figure 80 Role Manager Property Sheet



To view this **Property Sheet**, expand the **Station** folder and the **NiagaraNetwork NiagaraStationSys Def**, right-click **Views→AX Property Sheet** expand **RoleManager**.

Type	Value	Description
Retry Interval	hours minutes seconds (defaults to 15 seconds)	Defines the time to attempt to connect the server.
Last Success	read-only	Reports the last successful attempt time.
Last Attempt	read-only	Reports the date and time of the last attempted execution.
Next Execute Time	time	Defines the next execute time
Actual Role	read-only	Defines actual role of the user
Desired Role	drop-down list	Defines the roles that can be selected for the user.

niagaraDriver-SyncTask

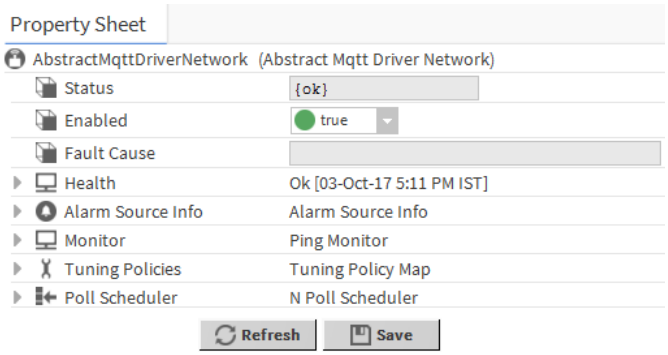
SyncTask is child container under the NiagaraStation's "Sys Def" device extension (NiagaraSysDefDeviceExt) that propagates changes to Sys Def components between stations that are in a "master/subordinate" relationship. The "Sys Def" feature is chiefly of interest to developers extending the API.

ndriver-NDeviceUxManager

This component is the top-level container component for an **AbstractMqttDriverNetwork** in a station.

This component is located in the **`abstractMqttDriver`** palette. Like other drivers, you should install this component under the **Drivers** node of a station. The default view of this component is the **Mqtt Client Driver Device Manager** view where you can add new devices, remove or edit devices in a tabular layout.

Figure 81 AbstractMqttDriverNetwork Property Sheet



To access these properties, expand **Config→Drivers**, right-click **AbstractMqttDriverNetwork** and click, **Views→AX Property Sheet**.

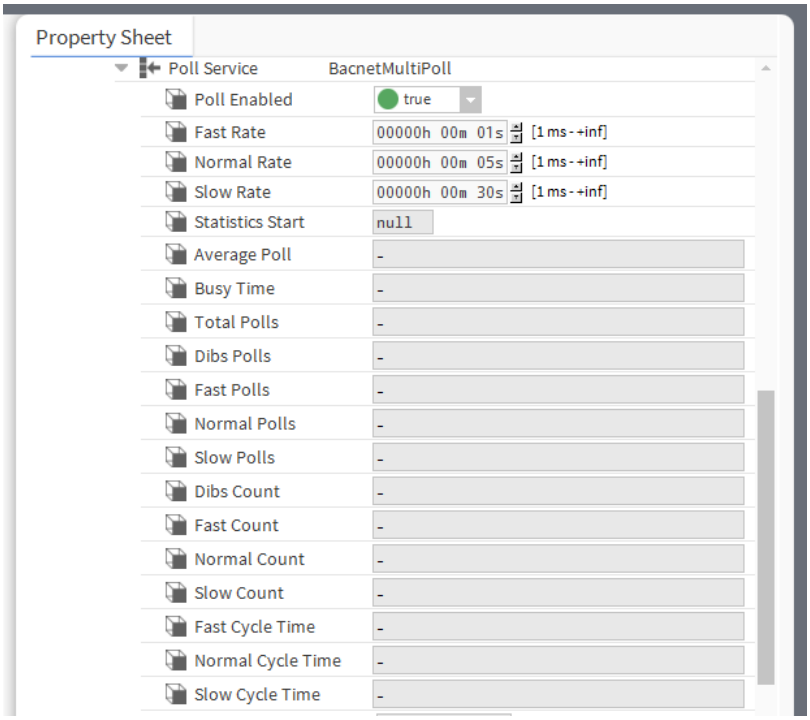
Property	Value	Description
Status	read-only	Reports the condition of the entity or process at last polling. `{ok}` indicates that the component is licensed and polling successfully. `{down}` indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. `{disabled}` indicates that the **`Enable`** property is set to `false`. `{fault}` indicates another problem. Refer to **`Fault Cause`** for more information.
Enabled	`true` or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Fault Cause	read-only	Indicates the reason why a system object (network, device, component, extension, etc.) is not working (in fault). This property is empty unless a fault exists.
Health	read-only	Reports the status of the network, device or component. This advisory information, including a time stamp, can help you recognize and troubleshoot problems but it provides no direct management controls. The *Niagara Drivers Guide* documents the these properties. These properties are documented in the *Niagara Drivers Guide*.
Alarm source info	additional properties	Contains a set of properties for configuring and routing alarms when this component is the alarm source. For property descriptions, refer to the *Niagara Alarms Guide* These properties are documented in the *Niagara Alarms Guide*.
Monitor	additional properties	Configures a network's ping mechanism, which verifies network health. This includes verifying the health of all connected objects (typically, devices) by pinging each device at a repeated interval.

Property	Value	Description
		The *Niagara Drivers Guide* documents these properties. These properties are documented in the *Niagara Drivers Guide*.
Tuning Policies	additional properties	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests. These properties are documented in the *Niagara Drivers Guide*.
Poll Scheduler	additional properties	Enables (`true`) and disables (`false`) polling. In history views: `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies). In networks: `true` polls all associated proxy points and devices that can be polled under the network component or, for a BacnetPoll, under that BacnetStack, Network, and Port. `false` suspends polling and further value updates from polling do not occur. **NOTE:** Poll Service actions Enable and Disable allow access to this property. These properties are documented in the *Niagara Drivers Guide*.

ndriver-NPollScheduler

Properties of the Poll Service component for typical field bus drivers include four writable properties and various read-only statistics properties. These properties appear in various **Property Sheet**s under a driver's network component.

Figure 82 Example of BACnet Ip Poll Service standard properties



You access this **Property Sheet** by expanding **Config→Drivers** followed by double-clicking or right-clicking the net network node, clicking **Views→Property Sheet** and expanding the **Poll Service** or **Poll Scheduler**.

The table documents the standard Poll Service properties each driver supports.

Property	Value	Description
Poll Enabled	`true` (default) or `false`	Enables (`true`) and disables (`false`) polling. In history views: `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies). In networks: `true` polls all associated proxy points and devices that can be polled under the network component or, for a BacnetPoll, under that BacnetStack, Network, and Port. `false` suspends polling and further value updates from polling do not occur. **NOTE:** Poll Service actions Enable and Disable allow access to this property.
Fast Rate	hours minutes seconds (defaults to 1 second)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.

Property	Value	Description
Normal Rate	hours minutes seconds (defaults to 5 seconds)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.
Slow Rate	hours minutes seconds (defaults to 30 seconds)	Sets the target polling interval for devices that can be polled (they are pollable) and assigned to the Fast Rate group.
Statistics Start	read-only timestamp	Reports either the last manual reset of poll statistics or, if statistics have not been reset, the first steady state time immediately following the last station restart.
Average Poll	read-only	Reports the average time spent during each poll event. This does not relate to the total time required to complete a poll cycle for any of the three rates. It is the time spent polling a given group of objects before pausing and switching to another group either using the same or a different poll rate.
Busy Time	read-only	Displays a percentage of time spent by the poll thread actually polling points using all poll rates. Includes (in parentheses) the ratio of time spent polling/total time since statistics were restarted. Given a small amount of time is spent transitioning among poll rates, and with the thread sleeping to evenly space out polling messages, it is unlikely to ever see **`Busy Time`** reach exactly 100%. However, any percentage above 95% indicates that the poll thread is basically spending all of its time actually polling. **NOTE:** In the case of the Poll Service for a BACnet network port, because two threads are used for polling, it is possible to see a **`Busy Time`** approaching 200%. In this case, divide **`Busy Time`** in half to get an average busy time for each thread.
Total Polls	read only	Reports the total number of polls conducted and the time spent waiting for polls to execute. This time is the same time indicated in the ratio of the **`Busy Time`** property. Typically, the total poll count indicates the number of times the PollService polled any object. It is not a running total of the actual poll cycles.
Dibs Polls	read-only	Reports the percentage and ratio of the number of DIBS polls versus total polls.
Fast Polls	read-only	Reports the number of polls made processing the fast queue.
Normal Polls	read-only	Reports the number of polls made processing the normal queue.
Slow Polls	read-only	Reports the number of polls made processing the slow queue.
Dibs Count	read-only	Reports the current and average number of components in the DIBS stack. (DIBS stands for Distributed Internet Backup System).
Fast Count	read-only	Reports the current and average number of components in the fast queue.
Normal Count	read-only	Reports the current and average number of components in the normal queue.

Property	Value	Description
Slow Count	read-only	Reports the current and average number of components in the slow queue.
Fast Cycle Time	read-only	Reports the average cycle time for the fast queue.
Normal Cycle Time	read-only	Reports the average cycle time for the normal queue.
Slow Cycle Time	read-only	Reports the average cycle time for the slow queue.

Example: Total Polls

You have two points assigned to a fast policy of 1 second, two points assigned to a normal policy of 5 seconds, and two points assigned to a slow policy of 10 seconds. When the statistics update every 10 seconds, you would expect the total polls to increment by 26, where:

Total polls = (2 fast/1 sec * 10 sec) + (2 normal/5sec * 10 sec) + (2 slow/10 sec * 10 sec)

Total polls = 20 fast + 4 normal + 2 slow

Total polls = 26

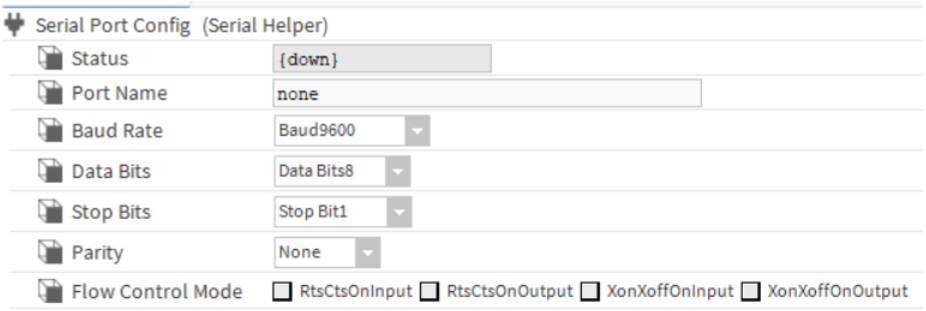
In some cases, such as a BacnetNetwork, **`Total Polls`** may indicate the number of poll messages sent to devices. Potentially, there could be multiple points being processed in a single message, such as if performing "read property multiple" for BACnet devices, or possibly when performing COV subscriptions.

Depending on the driver, additional statistics properties may be present in a poll component. For example, a BacnetPoll has additional properties **`Device Count`**, **`Point Count`**, and **`Object Count`**. See the specific driver guide for unique Poll Service features and information.

serial-SerialHelper

This container slot configures the serial port of any serial-based driver (found under the driver's network-level component as **Serial Port Config**).

Figure 83 SerialHelper properties



To access these properties, expand **Config→Drivers** and right—click the network node, click **Views→AX Property Sheet** and expand **Serial Port Config** or click **Serial Port Config**.

NOTE: These properties are also under any **SerialTunnel** (in a station's **TunnelService**). These components support a tunneling connection to the same type of network. You should configure these properties the same as for the SerialHelper of the associated driver network.

Property	Value	Description
Status	Read-only	Reports the condition of the entity or process at last polling. `{ok}` indicates that the component is licensed and polling successfully. `{down}` indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. `{disabled}` indicates that the **`Enable`** property is set to `false`. `{fault}` indicates another problem. Refer to **`Fault Cause`** for more information.
Port Name	text	Specifies the Port Name as `COM1` or `COM4` for connecting with the host platform.
Baud Rate	drop-down list (defaults to `Baud9600`)	Selects communication speed. Default value is `Baud9600`.
Data Bits	drop-down list (defaults to `Data Bits8`)	Selects the number of bits in a byte.
Stop Bits	drop-down list (defaults to `Stop Bit1`	Selects the number of stop bits to use.
Parity	drop-down list (defaults to `None`)	Selects the parity.
Flow Control Mode	check boxes	Selects the flow control. • `RtsCtsOnInput` • `RtsCtsOnOutput` • `XonXoffOnInput` • `XonXoffOnOutput` • `none` (default)

tunnel-TunnelService

This component is a station server for application tunneling, where remote PCs with a Niagara 4 Tunnel Client installed can use a legacy or vendor-specific PC application to access devices connected to one or more driver networks. A tunnel connection allows the remote client application to operate as it were directly attached to the driver network (via a virtual PC port).

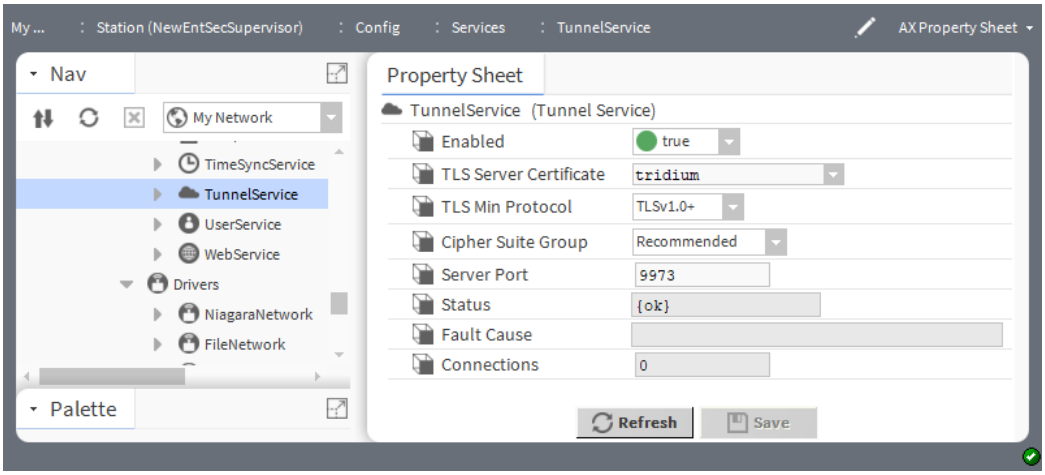
A client PC tunnels using an IP (LAN/WAN) connection, which is granted only after authentication as a station user (with admin write permissions for the particular child tunnel component to be accessed).

Currently, the following types of child tunnels are supported:

- SerialTunnel
- LonTunnel

In any station, only one **TunnelService** is recommended. It can hold the required number of child tunnels, as needed.

Figure 84 TunnelService properties



To access these properties, you must have added the **TunnelService** from the tunnel palette. Assuming you put this service in the **Services** container, expand **Config→Services** and double-click **TunnelService**.

In addition to several common properties (Enabled, Status, Fault Cause) the **TunnelService** contains the following configuration properties.

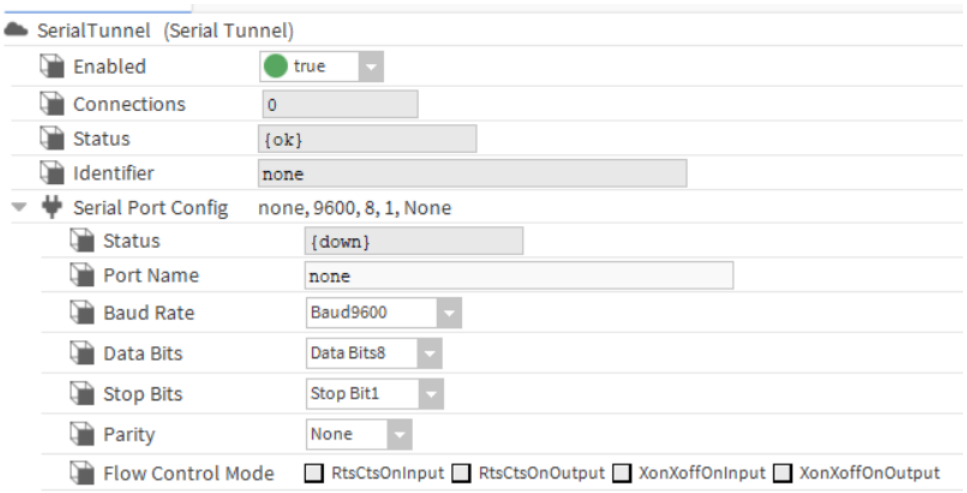
Property	Value	Description
TLS Server Certificate	drop-down list (defaults to `tridium`, which is a self-signed certificate)	Identifies the station's server certificate from the **User Key Store**. This certificate is password-protected by either a unique password or the global certificate password and should be signed by a root CA certificate. Do not rely for protection on a self-signed certificate.
Tls Min Protocol	drop-down list (defaults to `TLSv1.0+`)	Selects the earliest version of the TLS (Transport Layer Security) protocol supported by your network. This is the minimum level of the TLS. Options include versions `TLSv1.0+`, `TLSv1.1+`, `TLSv1.2+`, and `TLSv1.3`. Choosing a higher level provides more security. **NOTE:** As of Niagara 4.13, TLSv1.0 and TLSv1.1 are still supported for backwards compatibility, but it is recommended to use TLSv1.2 and higher. During the handshake, the server and client agree on which protocol to use. You should change this property from the default if your network requires a specific version or if a future vulnerability is found in one of the versions.
Cipher Suite Group	drop-down list (defaults to `Recommended`)	Controls which cipher suites can be used during TLS negotiation. The default is more secure than the other option (`Supported`) and should be used unless it causes compatibility issues with the client.
Server Port	number (defaults to 9973	Identifies the software port the driver monitors for incoming client tunnel connections.
Connections	read-only	Shows the number of active tunnel connections, which ranges from 0 (no active connections) to the number of child tunnel components.

tunnel-SerialTunnel

This component is the server-side component used to support tunneling of Windows serial-connected PC applications to devices reachable in a station's driver network. Typically, serial tunneling is used with a legacy vendor-specific PC program to access RS-232–connected devices attached to a controller.

You can add one or more **SerialTunnel**s under a station's **TunnelService**. Each **SerialTunnel** associates with one specific driver network (and corresponding controller serial port).

Figure 85 SerialTunnel properties



In addition to the standard property, Status, these properties are unique to this component.

Name	Value	Description
Enabled	`true` (default) or `false`	Turns serial tunneling on (`true`) and off (`false`).
Connections	read-only	Indicates the number of tunnel connections, as either 0 (none) or 1 (maximum).
Identifier	read-only	Reports the entered **`Port Name`** slot in the **Serial Port Config** container (below) used as the tunnel name when configuring the client-side **Serial Tunneling** window.
Serial Port Config (container)	additional properties	Holds configuration properties for the controller serial port as used by the specific driver network. Refer to the separate topic, **serial-SerialHelper** for property descriptions.

Actions

- **Disconnect All** disconnects any active connection through this **SerialTunnel** (maximum of 1). This removes the **TunnelConnection** below it. The remote (serial tunnel client) side displays a popup message: `Connection closed by remote host`.

  NOTE: Any **TunnelConnection** component also has its own **Disconnect** action, which effectively performs the same function.

tunnel-TunnelConnection

The driver dynamically adds this component under a tunnel component, such as a **SerialTunnel**, to reflect read-only information about this current tunnel connection.

To access this view, expand **TunnelService→SerialTunnel** and double-click **Tunnel Connection**.

Property	Value	Description
Established	read-only	Reports when this tunnel connection was first established.
User Name	read-only	Reports the user in the station that is currently tunneling.
Remote Host	read-only	Reports the Windows hostname or IP address of the remote tunneling client.
Protocol Version	read-only	Reports the version of the (remote) tunneling client application being used.
Last Read	read-only	Reports when the last read of a station item occurred over the tunnel connection.
Last Write	read-only	Reports when the last write to a station item occurred over the tunnel connection.

Actions

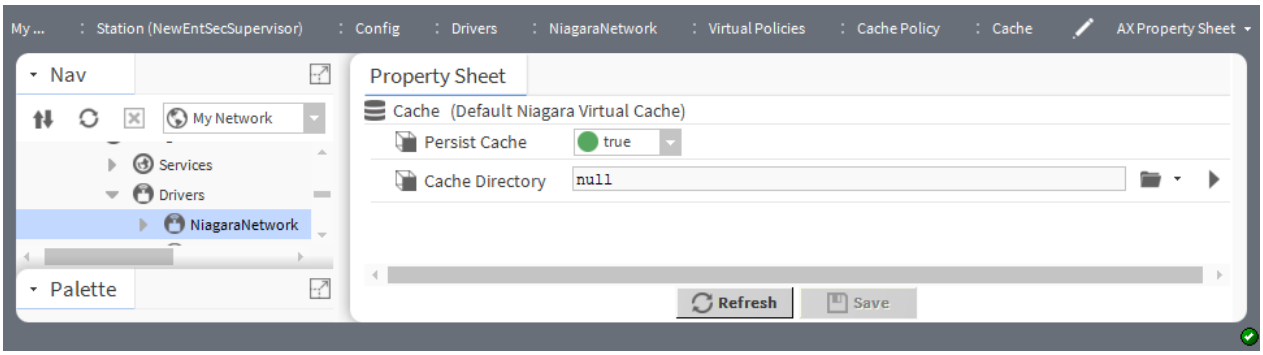
- **Disconnect All** disconnects any active connection through this **SerialTunnel** (maximum of 1). This removes the **TunnelConnection** below it. The remote (serial tunnel client) side displays a popup message: `Connection closed by remote host`.

  NOTE: Any **TunnelConnection** component also has its own **Disconnect** action, which effectively performs the same function.

niagaraVirtual-DefaultNiagaraVirtualCache

This component represents the **NiagaraNetwork**'s virtual component cache. It provides a default **Virtual Cache View** of persisted cached data (in file or files specified by the network's virtual **Cache Policy**).

Figure 86 Default Niagara Virtual Cache properties



To access these properties, expand **Config→Drivers**, right-click the **NiagaraNetwork** and click **Views→AX Property Sheet**, expand **Virtual PoliciesCache Policy**, right-click **Cache** and click **Views→AX Property Sheet**.

Property	Value	Description
Persist Cache	true (default), false	Slot data for virtual components is stored persistently in `cacheN.nva` file(s) in the specified "Cache Directory". Typically these file(s) are created upon station shutdown. If set to false, the virtual cache is not persisted.
Persist Cache	`true` default or `false`	Enables and disables cache for virtual components.
Cache Directory	Directory Chooser (defaults to `file:^niagaraDriver_nVirtual, null`)	Defines the ORD to the folder in which the driver creates one or more `cacheN.nva` file(s) when **`Persist Cache`** is set to true.

niagaraVirtual-NiagaraVirtualBooleanPoint

This component is a driver implementation of a virtual **BooleanPoint**, which are among the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualBooleanWritable

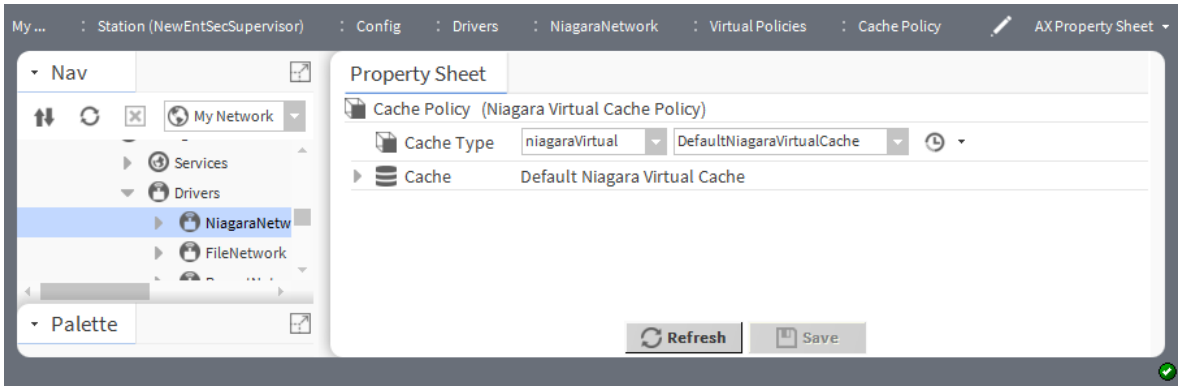
This component is a driver implementation of a virtual **BooleanWritable**, which is among the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualCachePolicy

This component (**Cache Policy**) configures the cache type for a **NiagaraNetwork**'s cache of virtual components.

NOTE: In most cases it is recommended you leave configuration properties at defaults.

Figure 87 Virtual Policies expanded in NiagaraNetwork property sheet



To access these properties, expand **Config→Drivers**, right-click the **NiagaraNetwork** and click **Views→AX Property Sheet**, expand **Virtual Policies**, and click **Cache Policy**.

Property	Value	Description
Cache Type	two drop-down lists (the first provides a single option `niagaraVirtual`; the second defaults to `DefaultNiagaraVirtualCache`)	Configures the type of cache. The second drop-down list has two options: `DefaultNiagaraVirtualCache` stores ORDs and associated data from accessing virtual components in memory. `NullCache` stores nothing in memory. In this case, virtual component access generates network traffic and slows the loading of graphics.
Cache	additional properties	Represents virtual cache data with the default **Niagara Virtual Cache View**. Its properties and view are documented in separate topics.

niagaraVirtual-NiagaraVirtualComponent

This component supports Baja virtual components (virtuals) in a Supervisor station. These components reside under the **NiagaraVirtualDeviceExt** (**Virtual** slot) of each **NiagaraStation**.

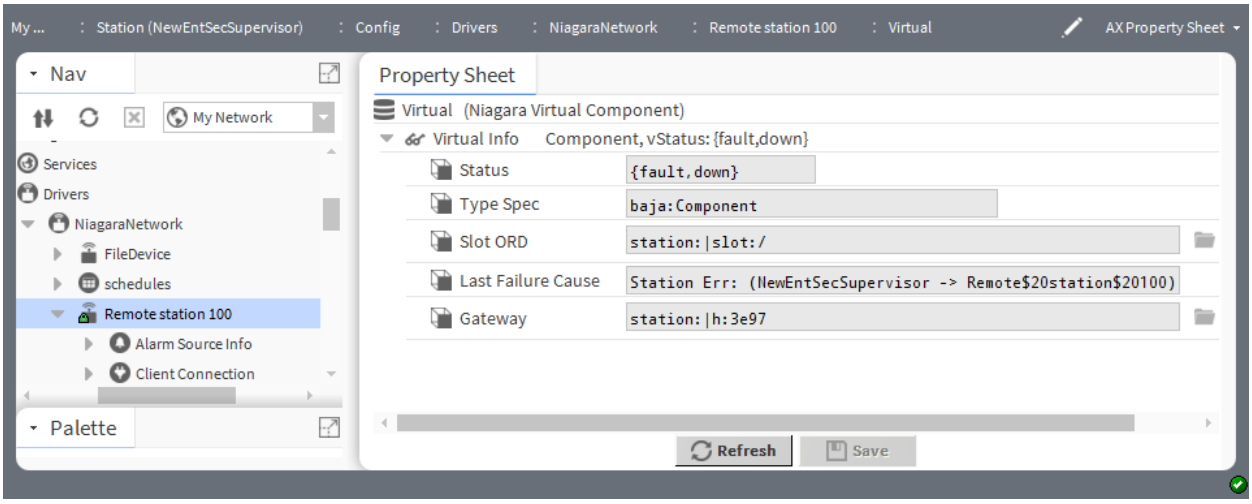
Typically, virtuals are possible only under a Supervisor station's **NiagaraNetwork** (the **`niagaraDriver`** feature in the host's license must have **`virtual`** attribute set to `true`). If the **`Virtuals Enabled`** property of the **NiagaraStation** is set to `true`, the entire component structure of that station can be dynamically modeled using virtuals.

The Workbench icon for each virtual has a small ghost superimposed in the lower right over the normal icon. This visual reminder indicates that you are looking at the virtual component space that represents the remote station's configuration.

The station provides only two meaningful views for virtual components: the virtual gateway's **Property** and **Category Sheets**. Special manager views for virtuals are not available. Regarding the other views and how they relate to virtuals:

- Wire Sheet: not available. The station does not support links to and from virtual components.
- Slot Sheet: available but of little practical application. Any changes made to slots (config flags, new slot, etc.) do not persist. However, this sheet displays the cryptic internal naming of virtual slots including the encoded facets and config flags used in ORD bindings from Px pages.
- Link Sheet: not available. The station does not support links to and from virtual components.

Figure 88 Virtual properties



To access these properties, expand **Config→Drivers→NiagaraNetwork→NiagaraStation** and double-click the **Virtual** node.

Property	Value	Description
Status (vStatus)	read-only	Reports the status of the virtual object. This is not to be confused with the status of the source component in the remote station. For example, a virtual for a proxy point may report `{ok}`, yet show an **Out** with a status of `{down}`. A virtual status may initially report `{stale}` before changing to `{ok}`.
Type Spec	read-only	Reports the `moduleName:componentType` used to represent the virtual, for example in the Property Sheet for the component.
Slot ORD	read-only	Reports the path to the remote source component relative to the remote station's slot hierarchy where the leading root portion of the path is `station:|slot:`.
Last Failure Cause	read-only	Reports a text string that indicates why the most recent attempt to execute failed.
Gateway	read-only	Identifies the station.

Refresh action

A virtual gateway only loads (activates) in memory the virtual components it needs to represent the remote station. Any one of these actions executes a driver-specific call to the device to gather these data. The results appear as child virtual components in the **NiagaraStation**:

- Double-clicking on the gateway's **Property Sheet**.
- Expanding the **NiagaraStation→Virtuals** node in the Nav tree.
- Viewing a Px page with widgets bound to virtuals under the **Virtual** gateway.

Viewing activated components places them in a memory cache for some minimum number of seconds. They remain as long as they are active (being viewed). After some maximum time of inactivity, the station automatically deletes the virtual components from this cache. By default, inactive cache life is about one (1) minute. In most cases, this default is fine. You can adjust cache settings for virtual components by changing entries in the host Supervisor's `!\lib\system.properties` file. This file includes helpful comments about these settings.

If you are engineering a job while configuration changes are still being made in a remote station (modeled by a **NiagaraStation** in the local **NiagaraNetwork**), you may need to use the right-click **Refresh** action on the virtual gateway.

As a general rule, this action is typically needed when the Nav tree contents for virtuals under a station do not display as expected.

niagaraVirtual-NiagaraVirtualDeviceExt

This component (**Virtual**) is a driver implementation of the Baja virtual gateway in the station. A virtual gateway is a component that resides under the station's component space (**Config**), and acts as a gateway to the station's virtual component space. Other object spaces are **Files** and **History**.

niagaraVirtual-NiagaraVirtualEnumPoint

This component is a driver implementation of a virtual **EnumPoint**, which is one of the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualEnumWritable

This component is a driver implementation of a virtual **EnumWritable**, which is one of the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualNumericPoint

This component is the **`niagaraDriver`** implementation of a virtual **NumericPoint**. This type of point is among the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualNumericWritable

NiagaraVirtualNumericWritable is the driver implementation of a virtual NumericWritable. They are among the eight type-specific virtual components. For more details, see "About virtual components".

niagaraVirtual-NiagaraVirtualStringPoint

This component is a driver implementation of a virtual **StringPoint**, which is one of the eight type-specific virtual components.

niagaraVirtual-NiagaraVirtualStringWritable

This component is a driver implementation of a virtual **StringWritable**, which is one of the eight type-specific virtual components.

Chapter 5 Plugins

Topics covered in this chapter

- Station Manager
- Delimited File Import Manager
- Device Manager
- Driver Manager
- File Device Manager
- Niagara History Export Manager
- Niagara History Import Manager
- Point Manager
- HTML5 Niagara Point Manager View
- About the Device Histories View
- Naxis Video Device Manager view
- HTML–5 Naxis Video Device Manager
- HTML-5 HTTP Client Device Manager
- HTML-5 Maxpro Device Manager
- HTML-5 NSnmp Device Manager
- HTML–5 Niagara Network Device Manager
- Niagara File Manager
- History Export Manager
- History Import Manager
- Device Histories View
- Niagara Point Manager
- Schedule Export Manager
- Schedule Import Manager
- Station Manager
- User Sync Manager
- Resource Manager
- Station Summary
- Server Connections Summary
- Niagara Virtual Cache View
- Nva File View

There are many ways to view plugins (views). One way is directly in the tree. In addition, you can right-click on an item and select one of its views. Plugins provide views of components.

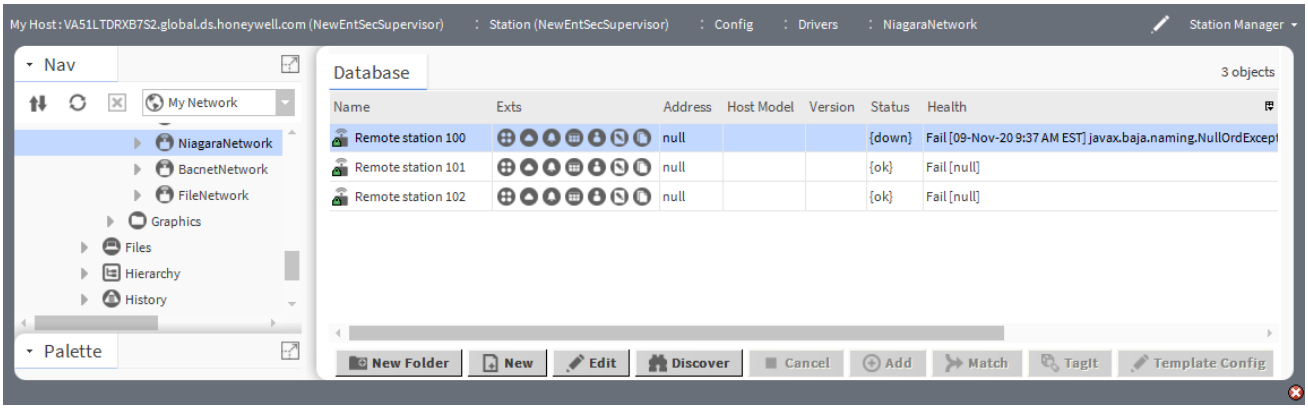
In Workbench, access the following summary descriptions on any plugin by selecting **Help→ On View** (F1) from the menu, or pressing F1 while the view is open.

Station Manager

This view manages the remote stations connected to a Supervisor station.

This is the default view of the **NiagaraNetwork**.

Figure 89 Station Manager view



To open this view, expand **Config→Drivers** and double-click the **NiagaraNetwork**.

Columns

Column	Description
Path	Reports the path to the remote station.
Name	Reports the name of the remote station.
Type	Reports the type of remote station.
Exts	Provides access to the extensions that are associated with the station. (Point Manager icon) opens the **Point Manager** view. (History Import Manager icon) opens the **Niagara History Import Manager** view. (Alarm icon) opens the **Alarm Device Ext Property Sheet**. (Schedule Import Manager icon) opens the **Niagara Schedule Import Manager** view. (User icon) opens the **Niagara User Device Ext Property Sheet**. (Sys Def icon) opens the **Niagara Sys Def Device Ext Property Sheet**. (File Manager icon) opens the **File Manager** view.
Address	Reports the IP address of the remote station.
Fox Port	Identifies the Fox Port in the remote station.
Use Foxs	Indicates if Fox communication in the remote station uses TLS security or not.
Host Model	Reports the hardware model of the remote controller.
Host Model Version	Reports the version of the hardware model.
Version	Reports the version of Niagara running on the remote controller.
Status	Indicates the current state of the remote station.
Enabled	Indicates if the remote station is enabled.
Health	Reports the current health of the remote station.
Fault Cause	If the remote station is in fault, reports the reason.
Client Conn	Reports the status of the client connection.

Column	Description
Server Conn	Reports the station of the server connection.
Virtuals Enables	Indicates if virtual entities are enabled in the remote station.

Buttons

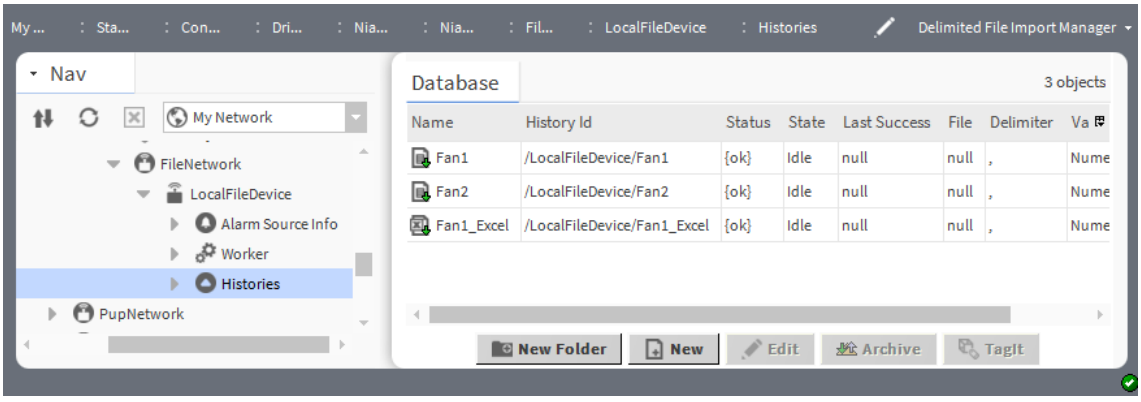
- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.
- **Template Config** accesses the station template that defines configuration options. You would select a template to set up the device with pre-configured properties.

Delimited File Import Manager

This view is the default view on the **Histories** extension (**FileHistoryDeviceExt**) of a **FileDevice** under a **FileNetwork**. This is the most-used view within a **FileNetwork** (outside of property sheets for created history file import descriptors)

This manager view is similar to other driver's history import managers. However, there is no **Discover** feature to add history file import descriptors (types `ExcelCsvFileImport` and `DelimitedFileImport`). Instead, you use the History Import **New** button to add history file import descriptors. Each is a reference a local delimited-type text file (such as a CSV type).

Figure 90 Delimited File Import Manager view



To open this view, expand **Config→Drivers→FileNetwork→FileDevice** and double-click the **Histories** node.

Columns

Column	Description
Name	Reports the name of the file.
History Id	Reports the full path to the file in the station.
Execution Time	Reports when the file import of this file occurred.
Enabled	Indicates if file import is enabled or disabled.
State	Indicates the current state of the data transmission.
Last Attempt	Displays the date and time when the driver last attempted to import this file.
Last Success	Displays the date and time when the driver last successfully imported this file.
Last Failure	Displays the date and time when the driver last failed to import this file.
Fault Cause	Indicates why the file failed to import.
Capacity	Displays the maximum number of history records allowed in the database.
Full Policy	Indicates what happens if the history runs out of room to store additional history records.
Interval	Reports a period of time.
Value Facets	Displays the configured facets.
Time Zone	Displays the time zone.
File	Displays information about the file.
Full Import on Execute	Displays how much of history to refresh on each import. `Enabled` imports the entire history on each import. `Disabled` appends only new data to the history on each import.
Row Start	Displays the row (record) in the file from which to begin importing data
Row End	Displays on which row (record) in the file to stop importing data.
Delimiter	Displays the text character the file uses to separate columns.
Timestamp Column Index	Displays a left-to-right index (zero-based) to identify the column in the file from which to import the timestamp.
Timestamp Format	Displays the format for timestamp.
Value Format	Displays the data type of history record.
Status Column Index	Displays the left-to-right index (zero-based) to identify the column in the file used to import the status, if available.
Identifier Column Index	Displays the left-to-right index (zero-based) to identify the column in the file used to filter rows for inclusion in combination with the **`Identifier Pattern`** property value).
Identifier Pattern	Displays the text string that the import search, where the import job imports any row with the matching text string.

Buttons

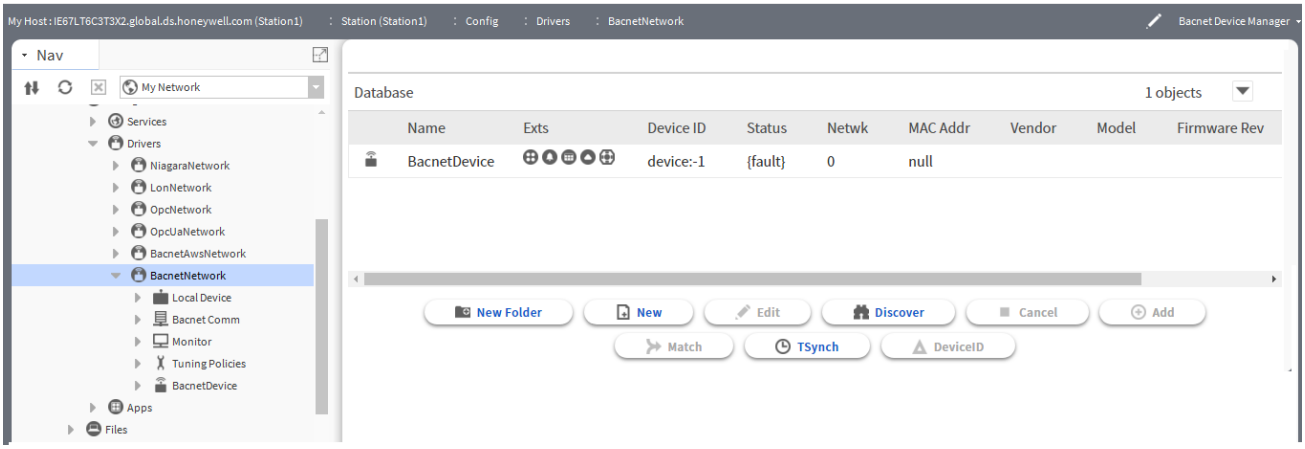
- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Device Manager

This is the default view for any network container (for a **NiagaraNetwork** only, this view is called the **Station Manager**). The **Device Manager** is a table-based view, where each row represents a unique device, as shown here. When building a network in the station, you use this view to create, edit, and delete device-level components.

Following station configuration, this view provides a status and configuration summary for all devices network using that driver. The Exts column provides quick double-click access to device extensions, such as Points for proxy points, Schedules for imported schedules, and so forth.

Figure 91 Example of a Device Manager (BacnetDeviceManager)



To open this view, expand **Config→Drivers** and double-click the network.

Columns

Column	Description
Name	Reports the name of the station.
Exts	Provides access to the extensions that are associated with the station. (Point Manager icon) opens the **Point Manager** view. (Alarm icon) opens the **Alarm Device Ext Property Sheet**. (Schedule Import Manager icon) opens the **Niagara Schedule Import Manager** view. (History Import Manager icon) opens the **Niagara History Import Manager** view. (Bacnet Config Manager icon) opens the **Bacnet Config Manager** view.
Device Id	Displays the Device Id.
Status	Indicates the current state of the remote station.
Enabled	Indicates if the remote station is enabled.
Health	Reports the current health of the remote station.
Network	Displays the network number.
MAC Addr	Displays the IP address to which it is connected.
Vendor	Displays the name of vendor using the device.
Model	Displays the model in use.

Column	Description
Firmware Rev	Displays the Firmware revision in use.
Segmentation	
App SW Version	Displays the software version of the application.
Encoding	

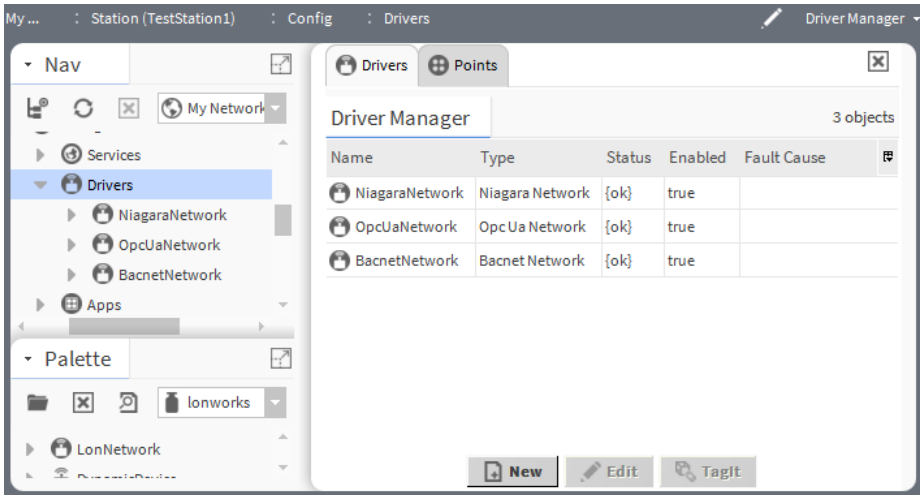
Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TSync** synchronizes the time.

Driver Manager

This view adds and displays driver records. It is available on the **Driver** container. The **Driver Manager** is its default view for all networks in a station.

Figure 92 Example of a Driver Manager



To open this view, expand **Config** and double-click the **Drivers**.

Columns

Column	Description
Name	Reports the name of the station.
Type	Reports the type of station.
Status	Indicates the current state of the remote station.

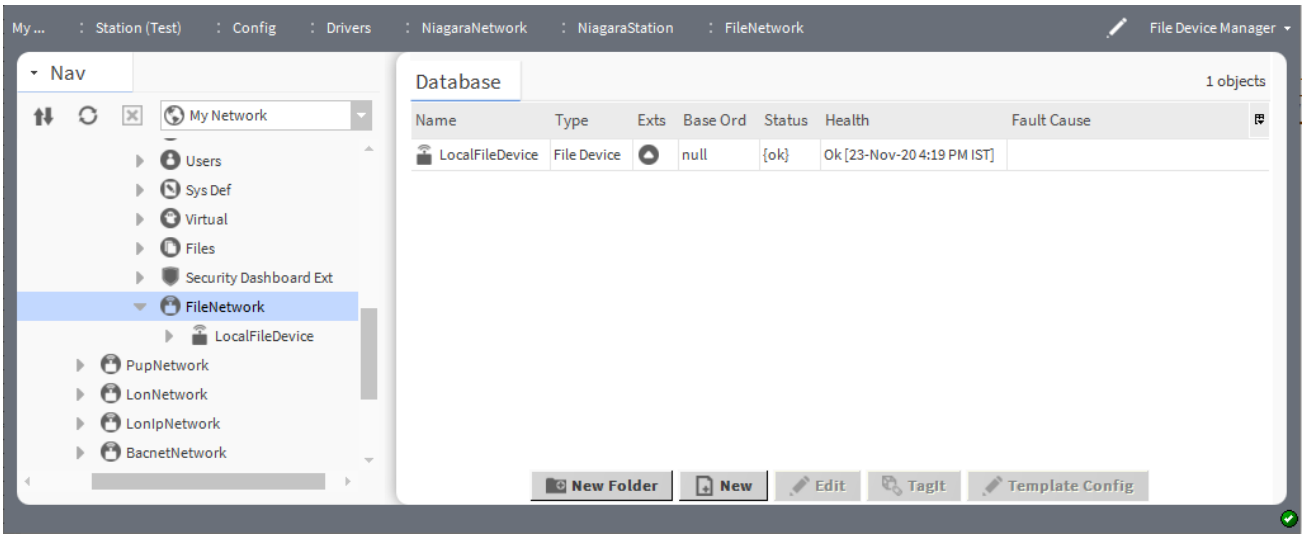
Column	Description
Enabled	Indicates if the remote station is enabled.
Fault Cause	If the remote station is in fault, reports the reason.

Buttons

- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **TagIt** associates metadata, such as location or unique configuration with the object.

File Device Manager

This is the default view on a **FileNetwork**.



To open this view, expand **Config→Drivers** and double-click the **FileNetwork** node in the Nav tree.

Columns

Column	Description
Name	Reports the name of the file.
Type	Reports the type file.
Exts	Provides access to a single extension (, History Import Manager icon) that opens the **Niagara History Import Manager** view.
Base Ord	Displays the path to the file.
Status	Indicates the current state of the file.
Enabled	Indicates if the station supports file-type devices.
Health	Reports the current health of the file.
Fault Cause	If the file in fault, reports the reason for the fault.

Buttons

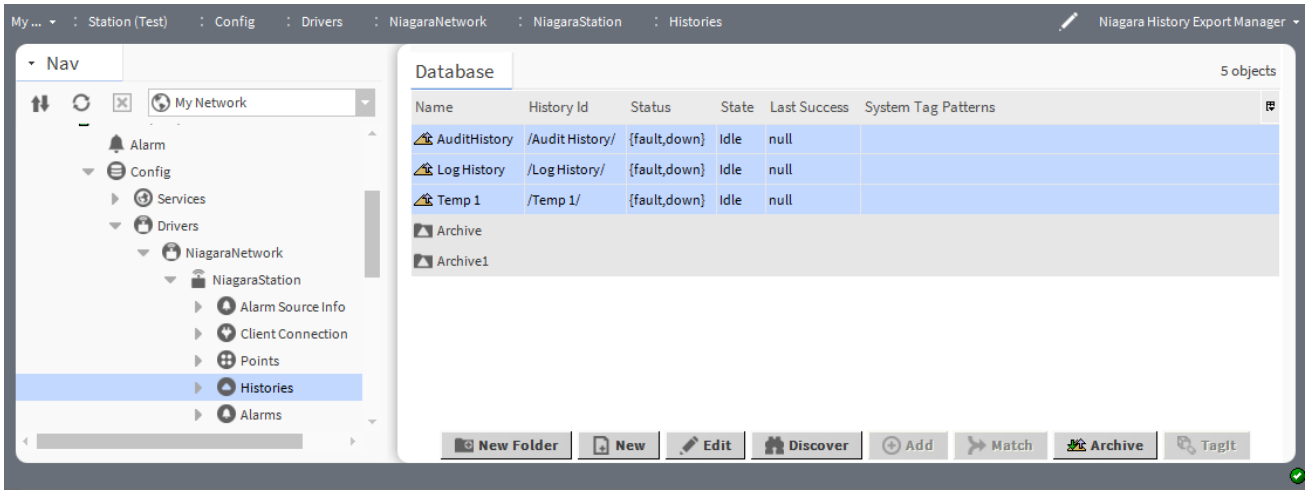
- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.

- **Edit** opens the device's database record for updating.
- **TagIt** associates metadata, such as location or unique configuration with the object.
- **Template Config** accesses the station template that defines configuration options. You would select a template to set up the device with pre-configured properties.

Niagara History Export Manager

This is the default view on the **NiagaraHistoryDeviceExt**.

Figure 93 Example of a Niagara History Export Manager



To open this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation** and right-click **Histories** and click **Views→Niagara History Export Manager**.

Columns

Column	Description
Name	Reports the history name.
History Id	Reports the history ID.
Execution Time	Reports when the export of this history occurred.
Enabled	Indicates if history export is enabled or disabled.
Status	Reports the condition of the exported history: `{ok}`, `{down}`, `{disabled}` or `{fault}`
State	Indicates the current state of the data transmission.
Last Attempt	Displays the date and time when the driver last attempted to export this history.
Last Success	Displays the date and time when the driver last successfully exported this history.
Last Failure	Displays the date and time when the driver last failed to export this history.
Fault Cause	Indicates why the history failed to export.
System Tag Patterns	Displays one or more tags used to filter the eportation of history records.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.

- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Niagara History Import Manager

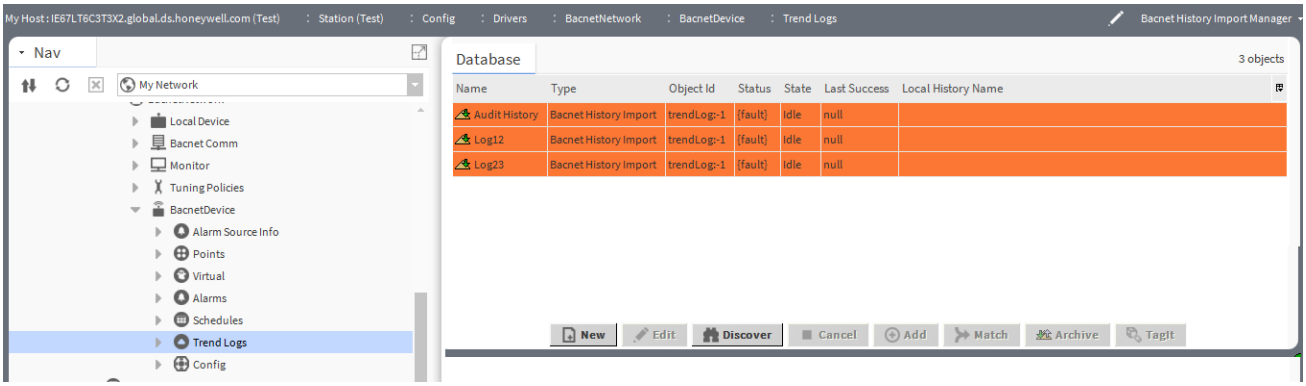
This is the default view for a device's **Histories** extension. Like other managers, it is a table-based view. Each row is a history import descriptor. Each descriptor specifies how log data are imported (pulled) from the device into the Supervisor station as histories.

You use this view to create, edit, and delete history import descriptors. Each import descriptor you add results in the creation of a local history—regardless if the source log data came from a BACnet Trend Log, history, or other type of data log (depending on driver type, and parent device's component type).

This view provides a status summary for collecting imported histories. You can also use it to issue manual **Archive** commands to one or more history descriptors. This causes an immediate import request to pull logged data from the remote device.

NOTE: Only history import descriptors appear in the **Niagara History Import Manager** view. Other components that may also reside under **Histories** do not appear. For example, you do not see the default **Retry Trigger** component. However, you can use the **Histories Property Sheet** to access these items.

Figure 94 Example of a History Import Manager under a BacnetDevice (Trend Logs)



To open this view, expand **Config→Drivers**, expand the driver and double-click the **Histories** node (**Trend Logs** in the example above).

Columns

The information in this view depends on the specific driver. These columns are available in the **NiagaraNetwork**'s **History Import Manager**.

Column	Description
Name	Reports the name of the descriptor (file, import or export).
History Id	Reports the history ID.
Execution Time	Reports when the import of this history occurs.
Enabled	Indicates if history import is enabled (`true`) or disabled (`false`).

Column	Description
Status	Reports the condition of the import descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Last Attempt	Reports the date and time of the last attempted import.
Last Success	Reports the date and time of the last successful import.
Last Failure	Reports the date and time of the last time the import job failed.
Fault Cause	Reports the reason for a failure.
Capacity	Reports the total number of history records allowed in the table.
Full Policy	Reports what happens when the table reaches capacity.
On Demand Poll Enabled	Reports the on-demand polling rate: `Fast`, `Normal` or `Slow`.
On Demand Poll Frequency	Indicates how frequently the station polls for history data.
System Tag Patterns	Reports one or more tag text strings used to filter imported records.

Buttons

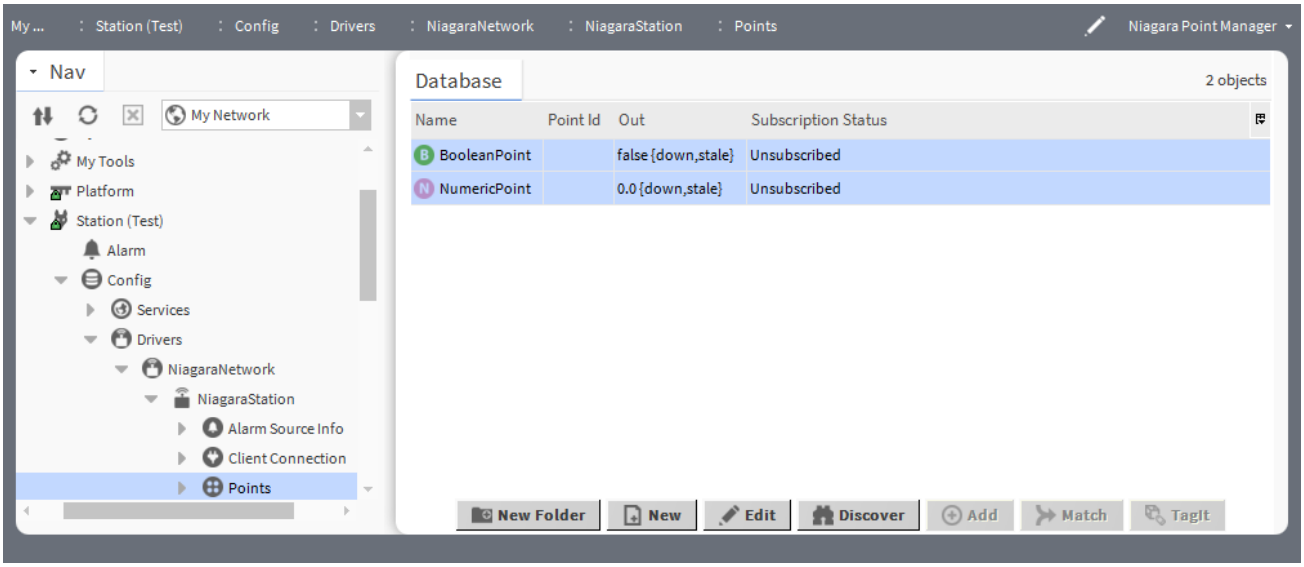
- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Archive** archives the imported history.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Point Manager

This view lists the proxy points in the point device extension under the **NiagaraNetwork**. The Points folder is the top parent container for real-time data values originating from a device.

Point Manager is a view on the **NiagaraPointDeviceExt** of the station.

Figure 95 Point Manager view



To open this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation** and double-click the **Points** folder or right-click **Points** and select **Views→Point Manager**.

The Supervisor station represents as proxy points the actual points in the remote station. The Supervisor station reads the value from each point in the remote station and writes it to each proxy point.

Columns

Column	Description
Path	Reports the path to the point.
Name	Reports the name of the point.
Type	Indicates the type of point
Display Name	Reports the display name for the point.
Point Id	Displays the point's ID.
Enabled	Indicates if the point is enabled or disabled.
Fault Cause	Reports why a point is in fault.
Facets	Indicates the unit of measure used for this point.
Tuning Policy Name	Displays the name of the network tuning policy currently in operation.
Subscription Status	Displays the status of point subscription.
Read Value	Displays the point's value.
Device Facets	Displays the unit of measure for the parent device for this point.
Conversion	Displays the type of units conversion.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.

- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

HTML5 Niagara Point Manager View

In Niagara 4.12 and later, there is added browser support for the Niagara 4 Point Manager View. The HTML5 Niagara Point Manager is a web-browser-based implementation of the Niagara 4 Point Manager. You can access it from the browser of your desktop or mobile device. It provides the same functions as the Workbench view.

To access this view when logged in to the Niagara 4 web interface via browser, expand **Config→Drivers→NiagaraNetwork**, right-click **Points** and select **Views→Niagara Point Manager**. To view the proxy point in the **Discovered** pane, click **Discover**. You can add the proxy points to the **NiagaraNetwork** database by clicking **Add**.

Figure 96 HTML5 Niagara Point Manager

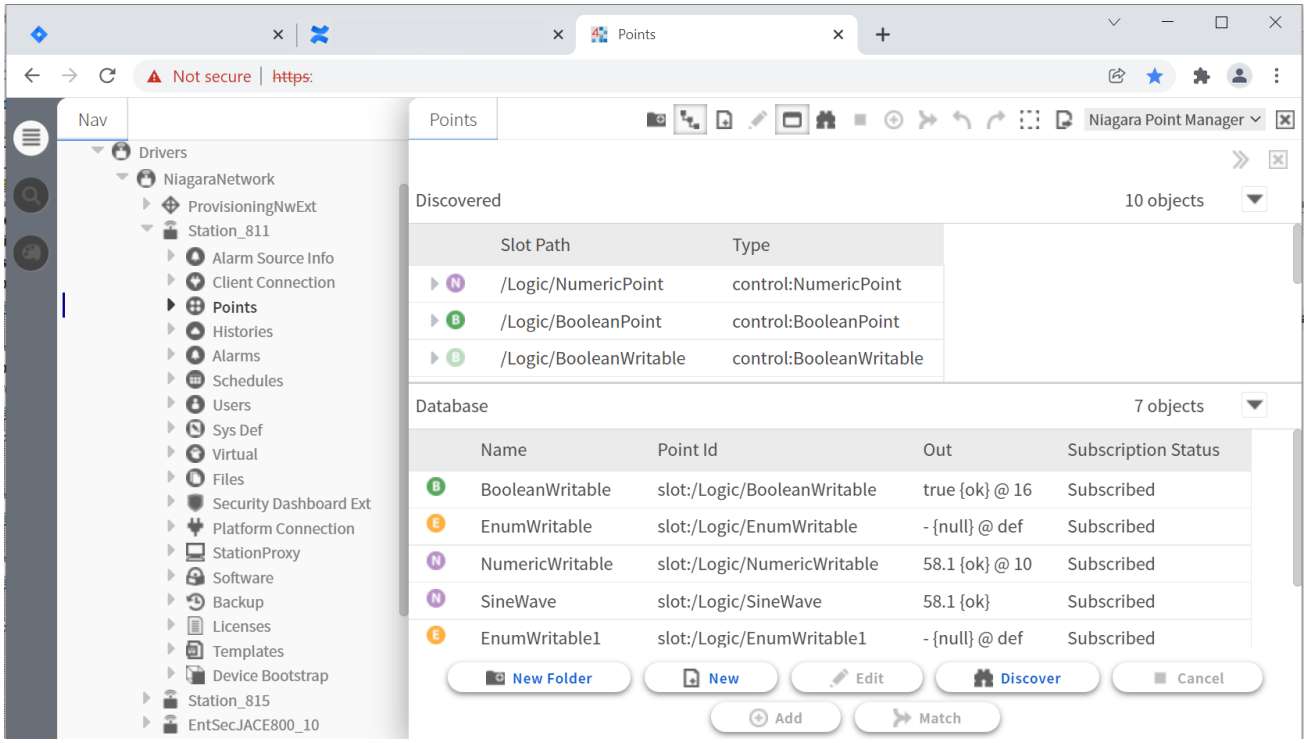
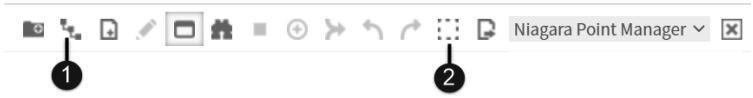


Figure 97 Toolbar of HTML5 Niagara Point Manager



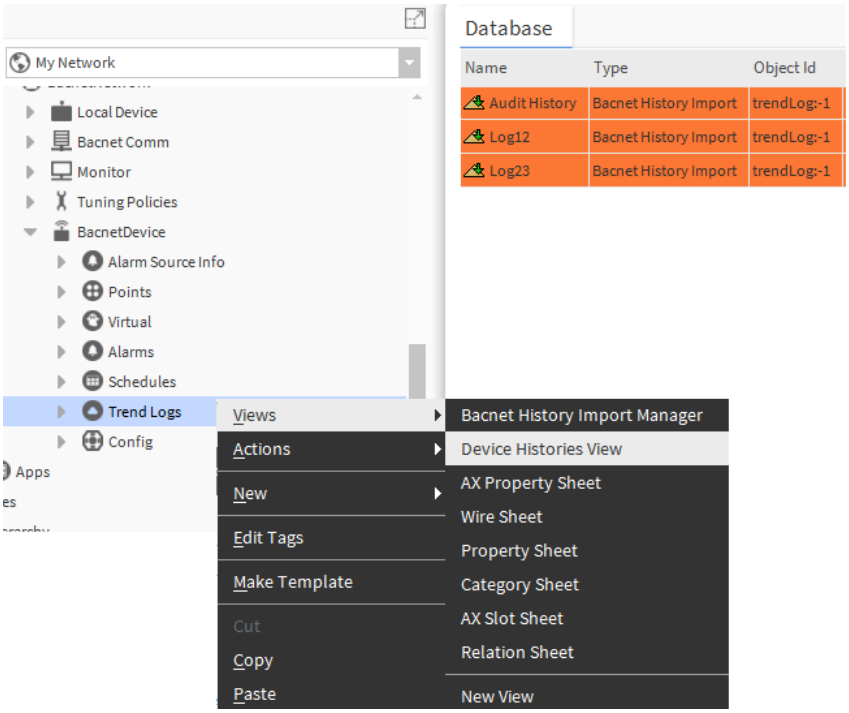
The toolbar offers you tools that you are familiar with from working in Workbench. The following tools, however, are unique to the HTML5 view.

1	Trace Descendants Displays all descendants or immediate children of the selected parent proxy point.
2	Multi-selection Mode Enables you to individually select multiple points without holding down the ctrl key.

About the Device Histories View

A **Device Histories View** is available on the Histories extension of any device that supports the import of histories (for example, BacnetDevice, NiagaraStation, and others), as well as receiving exported histories.

Figure 98 Device Histories View on the Histories (Trend Logs) extension of a BacnetDevice component



This view displays a filtered list of history shortcuts for histories imported or exported from this device, where shortcuts are automatically created by the view. If you double-click on a shortcut in the view, the default view of that local history displays. Right-click for a menu to select other history views.

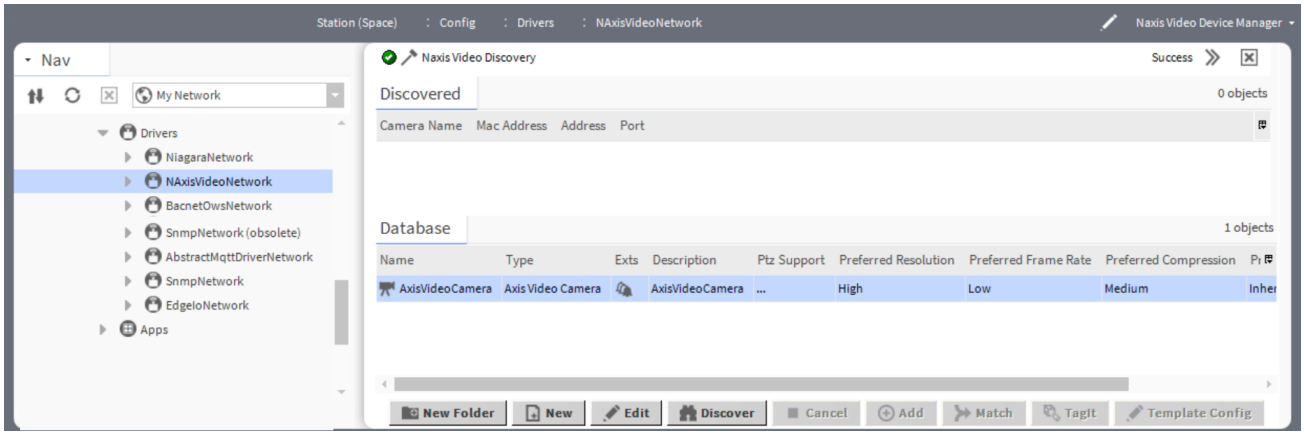
NOTE: The shortcut icon in the **Device Histories view** is a visual reminder that the view is displaying a shortcut, not the actual history.

In addition to this automatically-populated convenience view, you can add other history shortcuts anywhere under a device component, using a History Shortcuts component copied from the **histories** palette. This may also be useful for imported/exported histories of a **NiagaraStation**, when engineering a Supervisor station.

Naxis Video Device Manager view

This view is the default view of the **NaxisvideoNetwork**. It provides for online discovery for the cameras or manually adding cameras to manage the building access.

Figure 99 Naxis Video Device Manager view



To access these view, expand **Config→Drivers** and double-click **NAxisVideoNetwork** or right-click **NAxis-VideoNetwork** and click **Views→NaxisVideodeviceManager**.

Discovered pane

When you click **Discover** this manager goes to Learn mode, splits into two panes, and executes a discover job. Following are the columns in the discovered pane:

Column name	Description
Camera name	Displays the camera name.
Mac Address	
Address	Displays the address of the device.
Port	Defines the communication port.

Database Pane

Column name	Description
Name	Display the camera name
Type	Display the type of the camera.
Exts	Provides extension for Alarms.
Description	Display the description of the camera.
URL Address	
Ptz Support	
Preferred Resolution	Specifies the pixel resolution of each transmitted frame. Click drop-down and select **High**,**Medium** and **Low** to set the pixel resolution.
Preferred Frame Rate	Defines the speed of the video stream. Click drop-down and select **High**,**Medium** and **Low** to set the frame rate for video streaming.
Preferred Compression	Specifies what level of compression is used during live video streaming. The actual compression values for these relative settings are defined in the video device. Higher compression uses less bandwidth but negatively affects image quality. Options are **None**, **High**, **Medium** and **Low**.

Column name	Description
Preferred Video Stream Fox	Selects or declines the use of Fox Streaming. For more information, seePreferred Video Stream Fox/Video Stream Fox options.
Credentials	
Use Tcp Transport	
Use Rtsp Stream	
Rtsp Username	
Rtsp password	
Host Name	
Web Client Http Port	
Web Client Https Port	
Token Over Https	
Web Auth Schme	

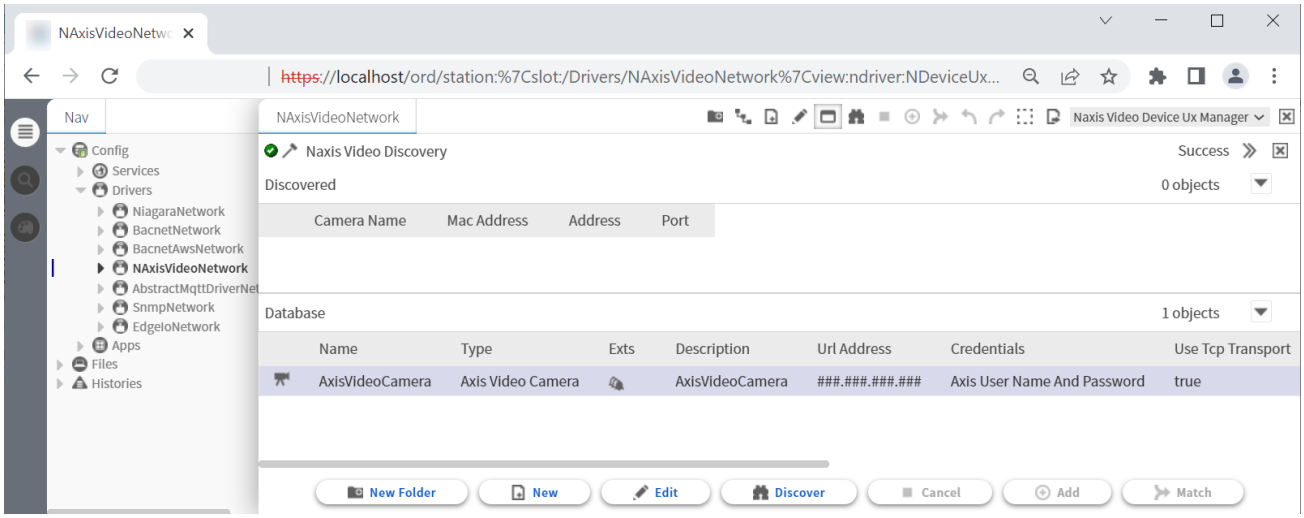
Buttons

-
-
-
-
-
-
-
- **TagIt** Associates an item of metadata, such as location or unique configuration, with the object.
- **Template Config** Accesses the station template that defines configuration options.

HTML–5 Naxis Video Device Manager

This view is a web-browser-based implementation of a **NAxisVideoNetwork**. It provides the same functions as the Workbench view.

Figure 100 Naxis Video Device UxManager view



You can access it from the browser and to access these view, expand **Config→Drivers** and double-click **NAxisVideoNetwork** or right-click the **NAxisVideoNetwork→Views→NaxisVideoDeviceUxManager**.

Discovered pane

When you click **Discover** this manager goes to Learn mode, splits into two panes, and executes a discover job. Following are the columns in the discovered pane:

Column name	Description
Camera name	Displays the camera name.
Mac Address	
Address	Displays the address of the device.
Port	Defines the communication port

Database Pane

Column name	Description
Name	Display the camera name
Type	Display the type of the camera.
Exts	Provides extension for Alarms.
Description	Display the description of the camera.
Url Address	
Credentials	
Use Tcp Transport	
Use Rtsp Stream	
Rtsp Username	
Rtsp password	
Host Name	

Column name	Description
Token Over Https	
Web Auth Schme	

Figure 101 Toolbar of Naxis Video Device UxManager view

Tools	Description
New Folder ( )	It creates a new folder in the database pane.
Trace Descendants ( )	Displays all descendants or immediate children of the selected parent proxy point.
Create new objects ( )	It creates new object in the database pane.
Edit objects ( )	Opens the device's database record for updating
Expand and collapse discovery pane ( )	Expands and collapses the objects in the database.
Start the discovery process ( )	Runs a discover job to locate installed devices, which appear in the Discovered pane. This view has a standard appearance that is similar to all **Device Manager** views.
Cancel the discovery process ( )	Ends the current discovery job.
Add the discovered object ( )	Inserts into the database a record for the discovered and selected object.
Match the discovered object ( )	Associates a discovered device with a record that is already in the database.
Undo ( )	Reverses the previous command.
Redo ( )	Restores a command–action after the Undo command has removed it.
Multi-selection Mode ( )	Enables you to individually select multiple points without holding down the ctrl key.
Export ( )	Exports the current view or object.

Buttons

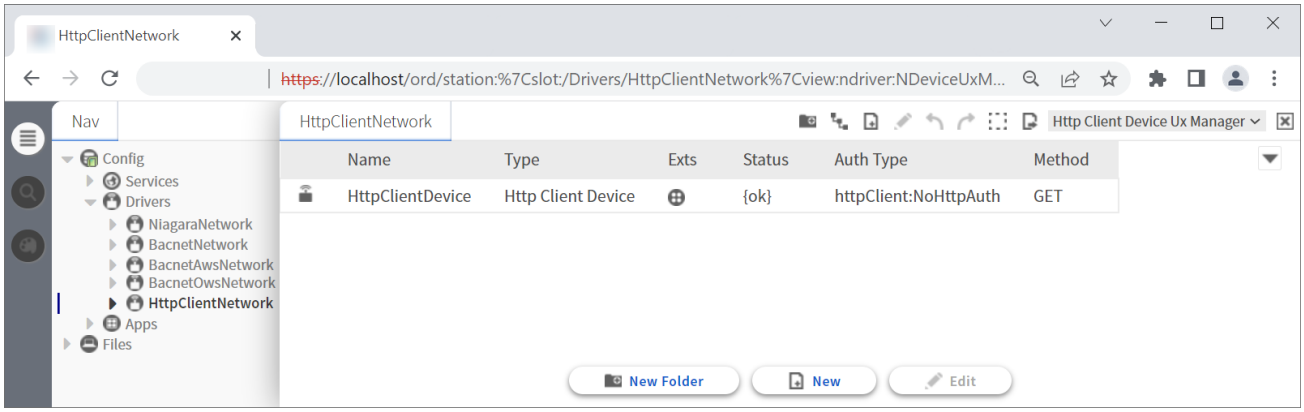
-
-
-
-
-
-

•

HTML-5 HTTP Client Device Manager

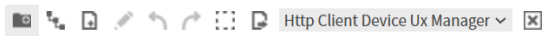
This view is a web-browser-based implementation of a **HTTPClientNetwork**. It provides same functions as Workbench view.

Figure 102 HTTP Client Device UxManager view



To access these view, expand **Config→Drivers** and double-click **HTTPClientNetwork** or right-click the **HTTPClientNetwork** and click **Views→HTTPClientDeviceUXManager**.

Figure 103 Toolbar of HTTP Client Device UxManager view



Tools	Description
New Folder ( )	It creates a new folder in the database pane.
Trace Descendants( )	Displays all descendants or immediate children of the selected parent proxy point.
Create new objects ( )	It creates new object in the database pane.
Edit objects ( )	Opens the device's database record for updating
Undo ( )	Reverses the previous command.
Redo ( )	Restores a command–action after the Undo command has removed it.
Multi-selection Mode ( )	Enables you to individually select multiple points without holding down the ctrl key.
Export ( )	Exports the current view or object.

Buttons

•

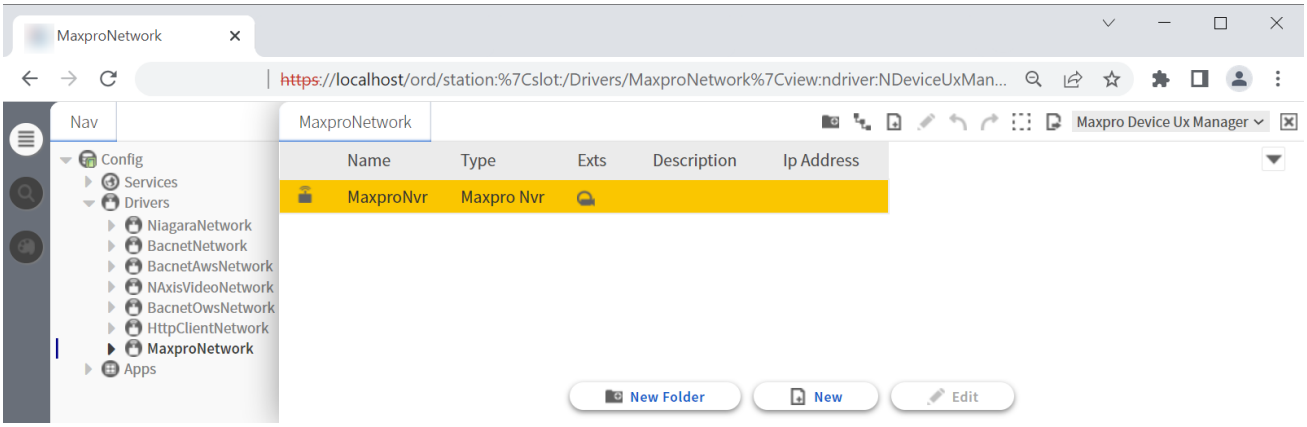
•

•

HTML-5 Maxpro Device Manager

This view is a web-browser-based implementation of a **MaxproNetwork**. You can access it from the browser where it provides same functions as Workbench view.

Figure 104 Maxpro Device UxManager view



To access these view, expand **Config→Drivers** and double-click **MaxproNetwork** or right-click the **MaxproNetwork** and click **ViewsMaxproDeviceUxManager**.

Figure 105 Toolbar of Maxpro Device UxManager view

Tools	Description
New Folder ( )	It creates a new folder in the database pane.
Trace Descendants ( )	Displays all descendants or immediate children of the selected parent proxy point.
Create new objects ( )	It creates new object in the database pane.
Undo ( )	Reverses the previous command.
Redo ( )	Restores a command–action after the Undo command has removed it.
Multi-selection Mode ( )	
Export ( )	Exports the current view or object.
Edit objects ( )	Opens the device's database record for updating

Buttons

-
-
-

HTML-5 NSnmp Device Manager

This view is a web-browser-based implementation of a **NSnmpNetwork**. It provides same functions as equivalent to Workbench view.

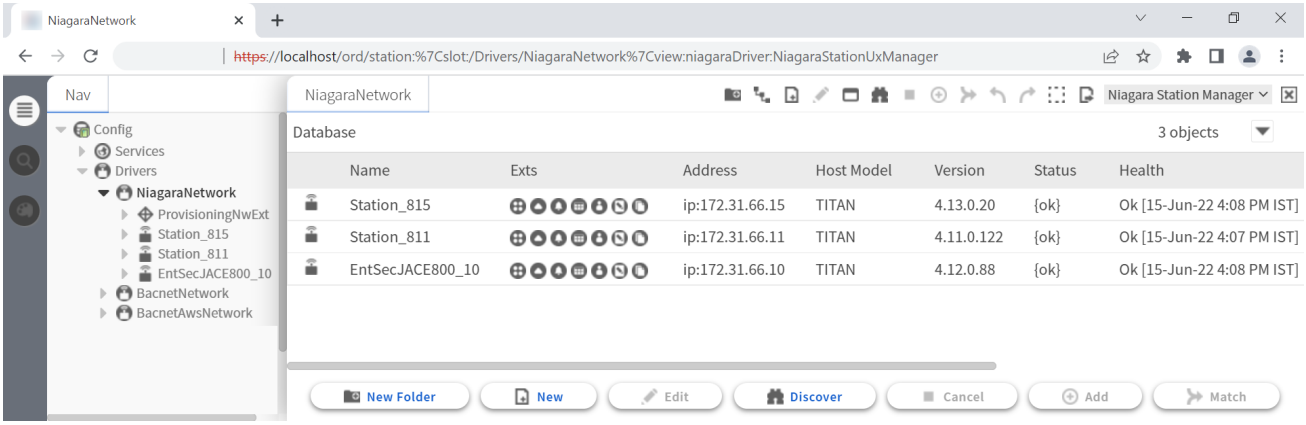
To access view from the browser, expand **Config→Drivers** and double-click **SnmpNetwork** or right-click **SnmpNetwork** and click **Views→NSnmpDeviceUxManager**.

HTML–5 Niagara Network Device Manager

In Niagara 4.13 and later there is added browser support for Niagara Network Device Manager. The HTML 5 **Niagara Station Manager** is a web-browser-based implementation of the **Station Manager** view. You can access it from a browser and It provides the same functions as the Workbench view.

The **DeviceManager** view is the default view of any network container, for a **NiagaraNetwork** only the view is called the **NiagaraStationManager**.

Figure 106 Niagara Station Manager view



To access this view when logged in to the Niagara 4 web interface via browser, expand **Config→Drivers** and double-click **NiagaraNetwork** (or) right-click **NiagaraNetwork→Views→NiagaraStationManager**.

Figure 107 Toolbar of Niagara Network Device Manager

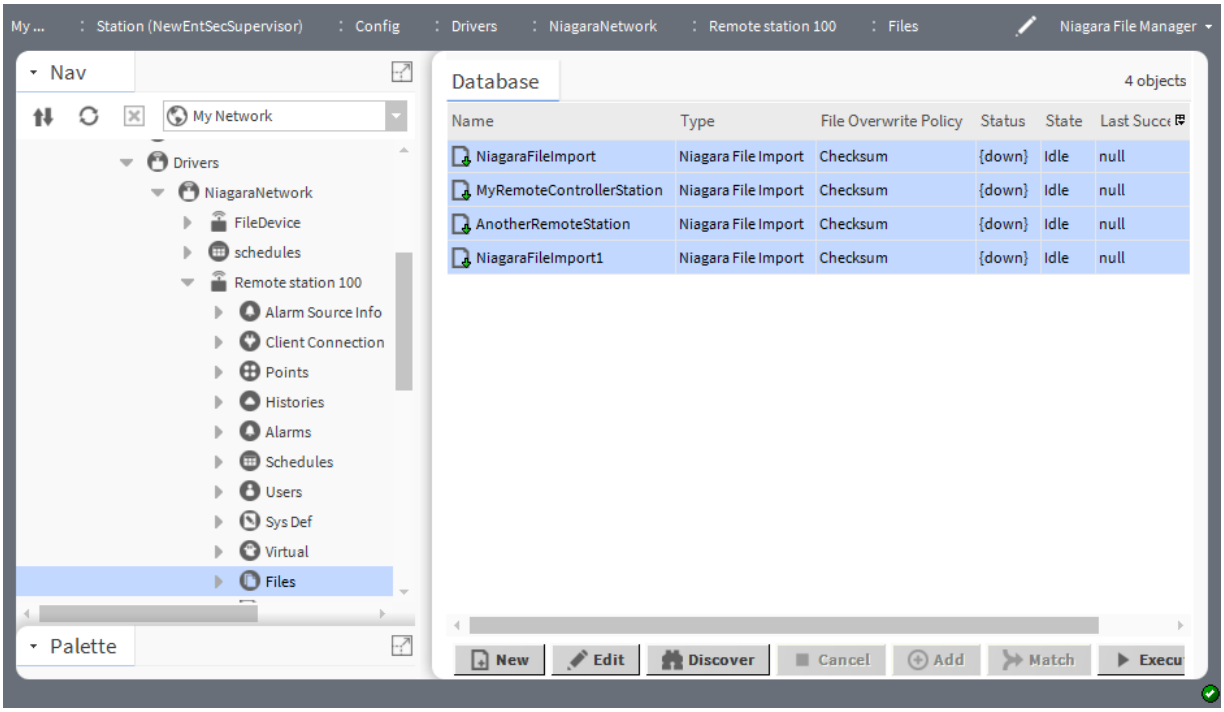
New Folder	It creates a new **NiagaraStationFolder** in the database pane.
Trace Descendants	Displays all descendants or immediate children of the selected parent proxy point.
Create new objects	It creates new object in the database pane.
Edit objects	Opens the device's database record for updating.
Expand and collapse discovery pane	Expands and collapses the objects in the database.
Start the discovery process	Runs a discover job to locate installed devices, which appear in the Discovered pane. This view has a standard appearance that is similar to all **Device Manager** views.
Cancel the discovery process	Ends the current discovery job.

Add the discovered object	Inserts into the database a record for the discovered and selected object.
Match	Associates a discovered device with a record that is already in the database.
Undo	Reverses the previous command.
Redo	Restores a command action after the Undo command has removed it.
Multi-selection Mode	Enables you to individually select multiple points without holding down the ctrl key.
Export	Exports the current view or object.

Niagara File Manager

Niagara File Manager is the default view of a **NiagaraStation**'s **Files** device extension (**NiagaraFileDeviceExt**) under a **NiagaraStation**.

Figure 108 Niagara File Manager view



To open this view, expand **Config→Drivers→NiagaraNetwork**, expand one of the **NiagaraStations** and double-click on the **Files** component, or right-click on the **Files** component and click **Views→Niagara File Manager**.

Columns

Column	Description
Name	Reports the name of the descriptor (file, import or export).
Type	Displays the type of file.
Execution Time	Reports when the import of this ile occurs.

Column	Description
Enabled	Reports if file import is enabled (`true`) or disabled (`false`).
File Overwrite Policy	Reports the criterion used to overwrite existing files upon any execution: `Checksum` or `Last Modified`.
Status	Reports the condition of the file import descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Last Attempt	Reports the date and time of the last attempted file import.
Last Success	Reports the date and time of the last successful file import.
Route To Station	As of Niagara 4.13, optionally specifies the route of intermediate station names (delimited by semi-colons) to reach the remote, reachable station from which to perform the file import operation. If this field is empty, the file is imported from the directly connected station.
Files	Reports the import target (`Local`) and source (`Remote`) file pair.
Last Failure	Reports the date and time of the last time the file import job failed.
Fault Cause	Reports the reason for a failure.

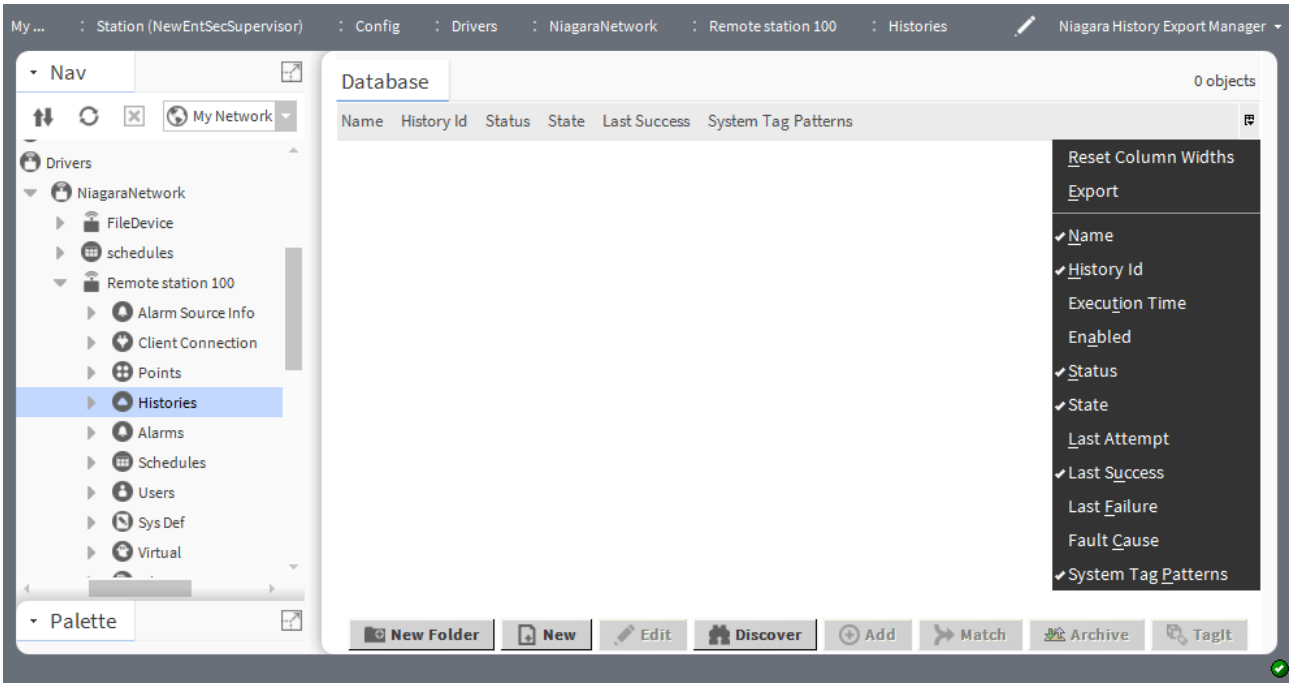
Buttons

- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **Execute** starts the importation of the selected files.

History Export Manager

This view is one of two views for the **NiagaraHistoryDeviceExt**.

Figure 109 Empty History Export Manager view



To open the view, right-click on a **NiagaraHistoryDeviceExt** in a **NiagaraStation** and select **Views→Niagara History Export Manager**.

Columns

The information in this view depends on the specific driver. These columns are available in the **NiagaraNetwork**'s **History Import Manager**.

Column	Description
Name	Reports the name of the descriptor (file, import or export).
History Id	Reports the history ID.
Execution Time	Reports when the export of this history occurs.
Enabled	Indicates if history export is enabled (`true`) or disabled (`false`).
Status	Reports the condition of the export descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Last Attempt	Reports the date and time of the last attempted export.
Last Success	Reports the date and time of the last successful export.
Last Failure	Reports the date and time of the last time the export job failed.
Fault Cause	Reports the reason for a failure.
System Tag Patterns	Reports one or more tag text strings used to filter exported records.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.

- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.
- **Archive** manually imports (pulls data) into one or more selected histories.

History Import Manager

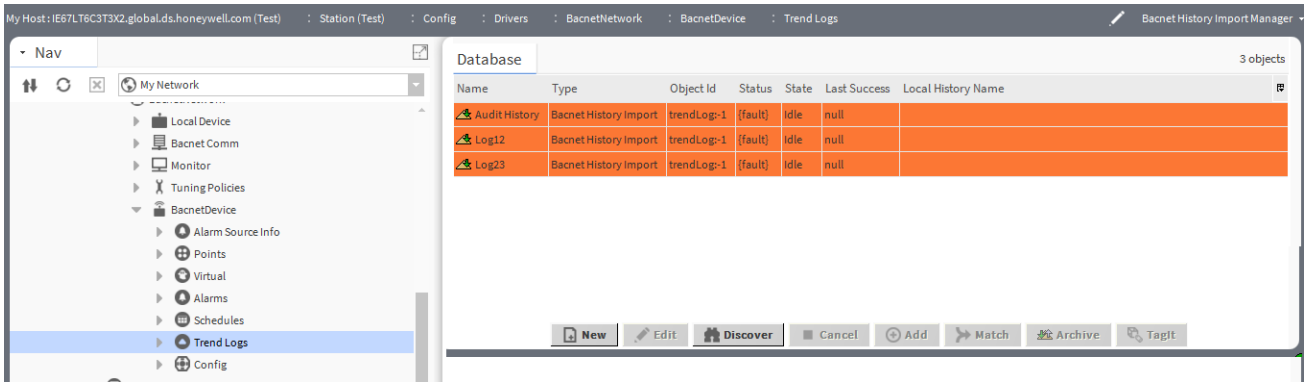
The **History Import Manager** is the default view for a device's **Histories** extension. Like other managers, it is a table-based view. Each row is a history import descriptor. Each descriptor specifies how log data are imported (pulled) from the device into the Supervisor station as a history.

You use these views to create, edit, and delete history import descriptors. Each import descriptor you add results in the creation of a local history—regardless if the source log data came from a BACnet Trend Log, history, or other type of data log (depending on driver type, and parent device's component type).

This view provides a status summary for collecting imported histories. You can also use it to issue manual **Archive** commands to one or more history descriptors. This causes an immediate import request to pull logged data from the remote device.

NOTE: Only history import descriptors appear in the **History Import Manager** view. Other components that may also reside under **Histories** do not appear. For example, you do not see the default **Retry Trigger** component. However, you can use the **Histories Property Sheet** to access these items.

Figure 110 History Import Manager under a BacnetDevice (Trend Logs)



To open this view, expand **Config→Drivers**, expand the driver and double-click the histories node (**Trend Logs** in the example above).

Columns

The information in this view depends on the specific driver. These columns are available in the **NiagaraNetwork**'s **History Import Manager**.

Column	Description
Name	Reports the name of the descriptor (file, import or export).
History Id	Reports the history ID.
Execution Time	Reports when the import of this history occurs.
Enabled	Indicates if history import is enabled (`true`) or disabled (`false`).

Column	Description
Status	Reports the condition of the import descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Last Attempt	Reports the date and time of the last attempted import.
Last Success	Reports the date and time of the last successful import.
Last Failure	Reports the date and time of the last time the import job failed.
Fault Cause	Reports the reason for a failure.
Capacity	Reports the total number of history records allowed in the table.
Full Policy	Reports what happens when the table reaches capacity.
On Demand Poll Enabled	Reports the on-demand polling rate: `Fast`, `Normal` or `Slow`.
On Demand Poll Frequency	Indicates how frequently the station polls for history data.
System Tag Patterns	Reports one or more tag text strings used to filter imported records.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **Archive** imports (pulls data) into one or more selected histories.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Device Histories View

This view is available on any device that supports import and export of histories (for example, BACnet devices, and others). The view shows a filtered list of history shortcuts for the particular device.

The view displays all related shortcuts in a table. You can double-click on any single entry in the table to open that history in the **Chart** view.

Niagara Point Manager

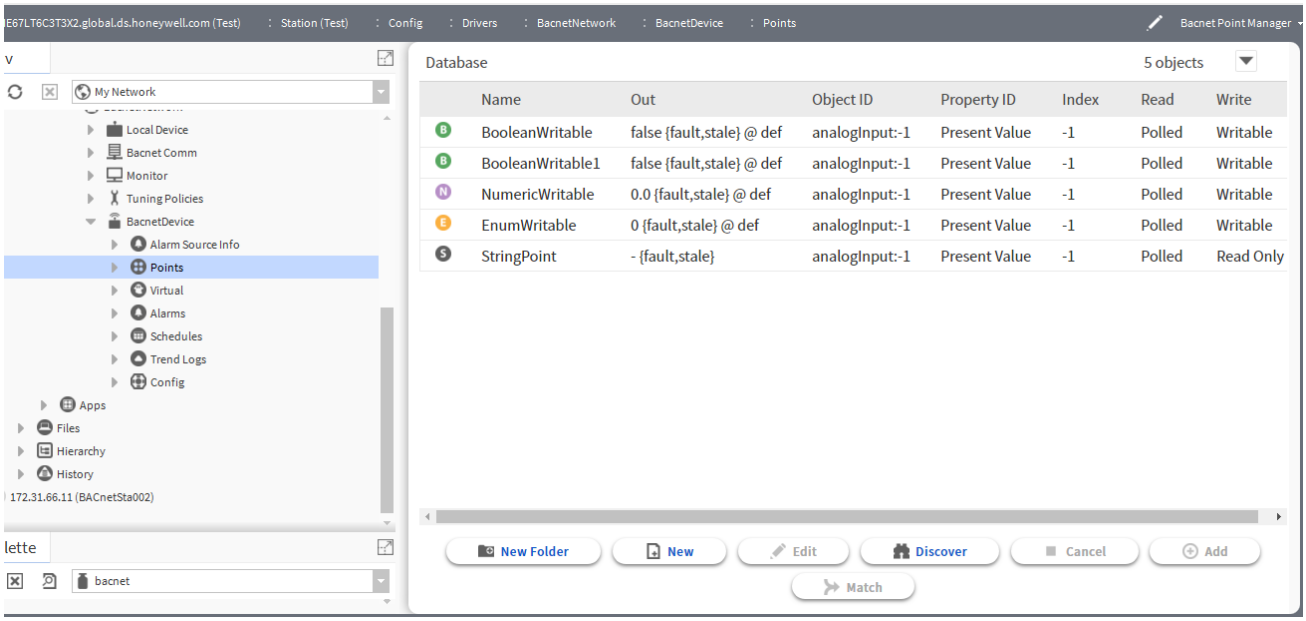
This manager provides access to the proxy points mapped into the **PointDeviceExt**. In a **NiagaraNetwork**, it is a view on the **NiagaraPointDeviceExt** of each **NiagaraStation**. It is the default view for the **Points** extension under any device object. Like other manager views, it is table-based. Each row represents a proxy point (or a points folder) under **Points**.

When building a network in the station, you use this view to create, edit, and delete proxy points in the station database.

Following station configuration, this view provides a status summary for proxy points. You can also use it to issue an override action to a writable proxy point, e.g. Active, Off, and so on.

NOTE: Only proxy points appear in the **Point Manager** view—any other components that may also reside under Points do not appear. For example, you do not see kitControl or schedule components, or any control point with a null proxy extension. However, you can use other views of Points (wire sheet, property sheet, slot sheet) to access these items.

Figure 111 Point Manager under a BacnetDevice



To open this view...

Columns

The columns depend on the driver. This table documents the columns in a **NiagaraNetwork Point Manager**.

Column	Description
Path	Identifies the location of the device or event in the station. For example, it defines the path to the resource in the web service, that is, the path after the host address.
Name	Provides descriptive text that reflects the identity of the entity or logical grouping.
Display Name	Displays the name for the point.
Type	Displays the type of point.
Point Id	Displays the Id given to the point.
Out	Displays the current value of the proxy point including facets and status. The value depends on the type of control point. Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level. Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as `fault`, `overridden`, `alarm`, and so on. If no status flag is set, status is considered normal and reports `{ok}`.
Enabled	Reports if the point is functional.
Facets	Reports the facet setting.
Tuning Policy Name	Displays the selected tuning policy name.

Column	Description
Subscription Status	Displays the status of the subscription.
Fault Cause	Indicates the reason why a system object (network, device, component, extension, etc.) is not working properly (in fault). This property is empty unless a fault exists.
Read Value	Displays the last value read from the device, expressed in device facets.
Device Facets	Displays the configured facets.
Conversion	Displays the type of selected conversion.
Display Names	Displays the name for the points.

Buttons

These buttons are available on the **NiagaraNetwork**'s **Niagara Point Manager** view.

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

Schedule Export Manager

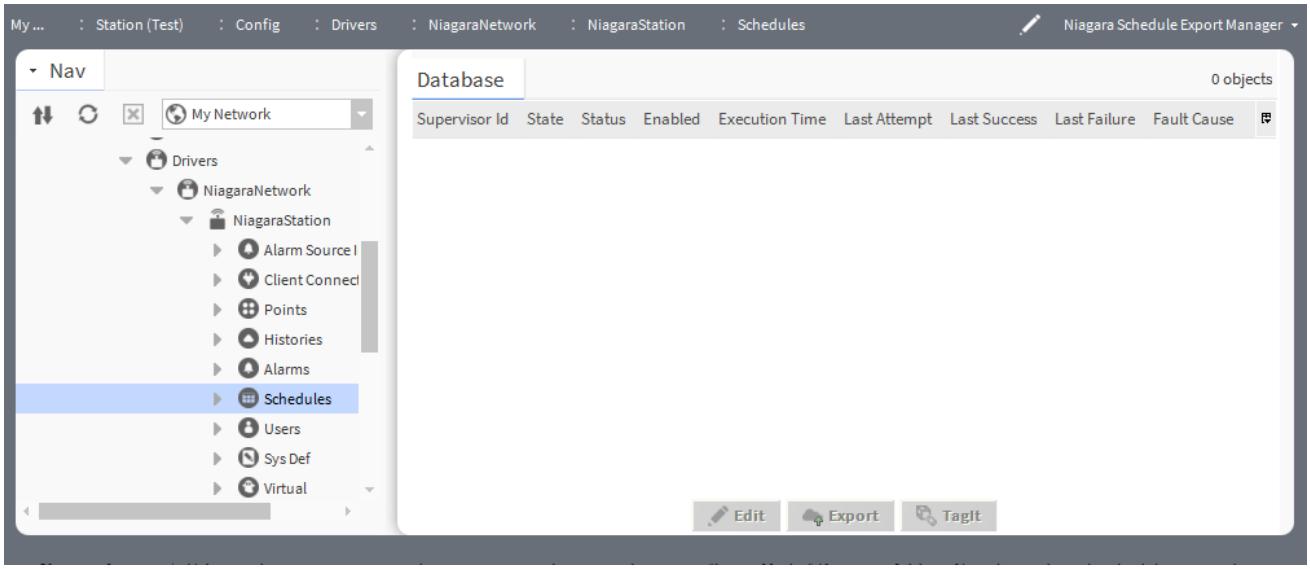
The Schedules extension of a NiagaraStation, ObixClient/R2ObixClient, or BacnetDevice has an available **Schedule Export Manager** view. It allows management of schedule components in the local station made available to that remote device.

This is the view on the **niagaraDriver-NiagaraScheduleDeviceExt** component. It manages schedules exported from a **NiagaraStation**.

Like other managers, this is a table-based view. Each row represents a schedule export descriptor. Each descriptor specifies how/when configuration for a local schedule is "pushed" to either:

- An imported schedule component in the designated **NiagaraStation**.
- An existing BACnet **Schedule** object or **Calendar** object in the designated **BacnetDevice**, or existing schedule in the designated oBIX server.

Figure 112 Schedule Export Manager under a NiagaraStation



To open this view, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, right-click **Schedules** and click **Views→Niagara Schedule Export Manager**.

A **Schedule Export Manager** works differently in **NiagaraNetwork**s from the way it works under a **BacnetNetwork** or **ObixNetwork**:

- **For a NiagaraStation**, you do not create export descriptors using this view—there is no Learn Mode, Discover, Add, or New. Instead, each schedule export descriptor is automatically created when the remote station imports a local schedule component.
- **For a BacnetDevice or ObixClient**, you do use Learn Mode to discover BACnet Schedule/Calendar objects or oBIX schedules in the device. Then, you select and add any as schedule export descriptor(s). In each export descriptor, you must specify the station's local schedule component that exports (writes) its event configuration to that remote schedule object.

After configuration, this view provides a status summary for exporting local schedules. You can also use it to issue manual Export commands to one or more schedules. This causes an export push of schedule configuration into the remote device.

NOTE: Only schedule export descriptors appear in the Schedule Export Manager view—any other components that may also reside under Schedules do not appear. For example, you do not see imported schedules or the default **Retry Trigger** component. However, the Nav tree and other views on Schedules provide you access to these items.

Columns

Column	Description
Supervisor Id	Displays the Supervisor ID.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Status	Reports the condition of the export descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
Enabled	Indicates if history export is enabled (`true`) or disabled (`false`).
Execution Time	Reports when the export of this history occurs.
Last Attempt	Reports the date and time of the last attempted export.

Column	Description
Last Success	Reports the date and time of the last successful export.
Last Failure	Reports the date and time of the last time the export job failed.
Fault Cause	Reports the reason for a failure.

Buttons

These buttons are available on the **NiagaraNetwork**'s **Niagara Point Manager** view.

- **Edit** opens the device's database record for updating.
- **Export** exports the schedule to specified format (pdf,csv).
- **TagIt** associates metadata, such as location or unique configuration with the object.

Schedule Import Manager

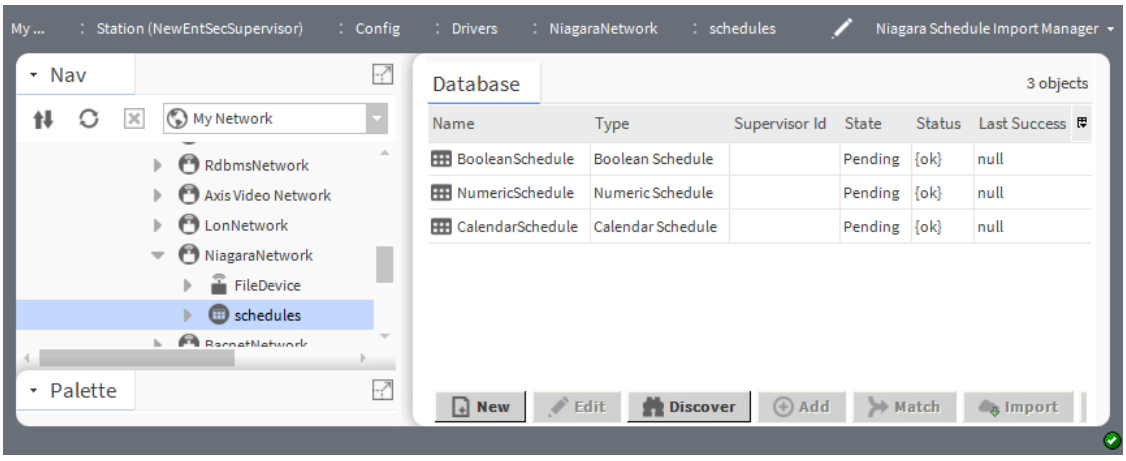
This manager is the default view for a device's Schedules extension (**niagaraDriver-NiagaraScheduleDeviceExt**). Like other managers, it is a table-based view, as shown here. Each row corresponds to an imported schedule (read-only). Configuration for each includes its name and whether it is enabled. It manages schedules imported from a NiagaraStation

When building a network in the station, you use this view to create, edit, and delete imported Niagara schedules. In the case of a NiagaraNetwork (only), each schedule that you import results in the creation of a remote "schedule export descriptor" in that remote station.

Following station configuration, this view provides a status summary for collecting imported schedules. You can also use it to issue manual **Import** commands to one or more schedules. This causes an immediate import request to pull schedule configuration data from the remote device.

NOTE: Only imported schedules appear in the **Schedule Import Manager**—any other components that may also reside under Schedules do not appear. For example, you do not see the default Retry Trigger component (see "About the Retry Trigger"), or if a NiagaraStation, schedule export descriptors. However, the Nav tree and other views on Schedules provide you access to these items.

Figure 113 Schedule Import Manager under a NiagaraStation



To open this view, expand **Config→Drivers→NiagaraNetwork**, expand a **NiagaraStation** and double-click **Schedules**.

Columns

The information in this view depends on the specific driver. These columns are available in the **NiagaraNetwork**'s **Niagara Schedule Import Manager**.

Column	Description
Name	Reports the name of the descriptor (file, import or export).
Type	Specifies the schedule type.
Supervisor Id	Displays the Supervisor ID.
Status	Reports the condition of the import descriptor. Usually `{ok}` unless **`Enabled`** is `false`, in which case it reports `{disabled}`. If there is a problem it reports `{false}` with a reason in the **`Fault Cause`**.
State	Reports the current state of the data transfer as: `Idle`, `Pending` or `In Progress`.
Enabled	Indicates if the network, device, point or component is active or inactive.
Execution time	Indicates if history import is enabled (`true`) or disabled (`false`).
Last Attempt	Reports the date and time of the last attempted import.
Last Success	Reports the date and time of the last successful import.
Last Failure	Reports the date and time of the last time the import job failed.
Fault Cause	Reports the reason for a failure.

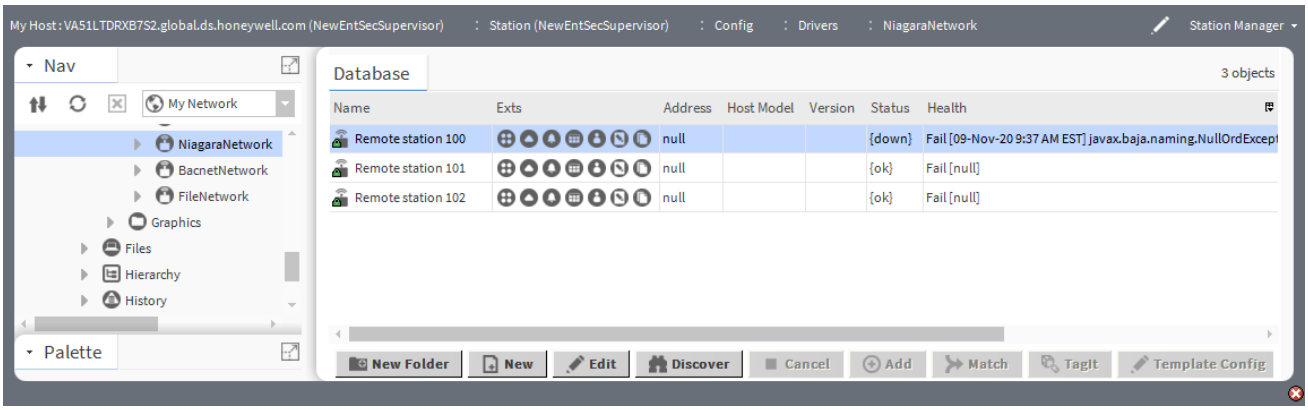
Buttons

- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.
- **Import** manually imports (pulls data) into one or more selected imported schedules.

Station Manager

This view on the **NiagaraNetwork** manages access to **NiagaraStation**s. It operates like the **Device Manager** for most drivers that have online device discovery capability.

Figure 114 Station Manager view



To open this view, expand **Config→Drivers** and double-click **NiagaraNetwork**.

Columns

Column	Description
Path	Reports the path to the station.
Name	Reports the name of the station.
Type	Reports the type of station.
Exts	Provides access to the extensions that are associated with the station.  (Point Manager icon) opens the **Point Manager** view.  (History Import Manager icon) opens the **Niagara History Import Manager** view.  (Alarm icon) opens the **Alarm Device Ext Property Sheet**.  (Schedule Import Manager icon) opens the **Niagara Schedule Import Manager** view.  (User icon) opens the **Niagara User Device Ext Property Sheet**.  (Sys Def icon) opens the **Niagara Sys Def Device Ext Property Sheet**.  (File Manager icon) opens the **File Manager** view.
Address	Reports the IP address of the remote station.
Fox Port	Identifies the Fox Port in the remote station.
Use Foxs	Indicates if Fox communication in the remote station uses TLS security or not.
Host Model	Reports the hardware model of the remote controller.
Host Model Version	Reports the version of the hardware model.
Version	Reports the version of Niagara running on the remote controller.
Status	Indicates the current state of the remote station.
Enabled	Indicates if the remote station is enabled.
Health	Reports the current health of the remote station.
Fault Cause	If the remote station is in fault, reports the reason.
Client Conn	Indicates if a client is connected.
Server Conn	Indicates if a server is connected.
Virtuals Enabled	Indicates if virtual entities are enabled in the station.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Discover** runs a discover job to locate installed devices, which appear in the **Discovered** pane. This view has a standard appearance that is similar to all **Device Manager** views.
- **Cancel** ends the current discovery job.
- **Add** inserts into the database a record for the discovered and selected object.
- **Match** associates a discovered device with a record that is already in the database.
- **TagIt** associates metadata, such as location or unique configuration with the object.

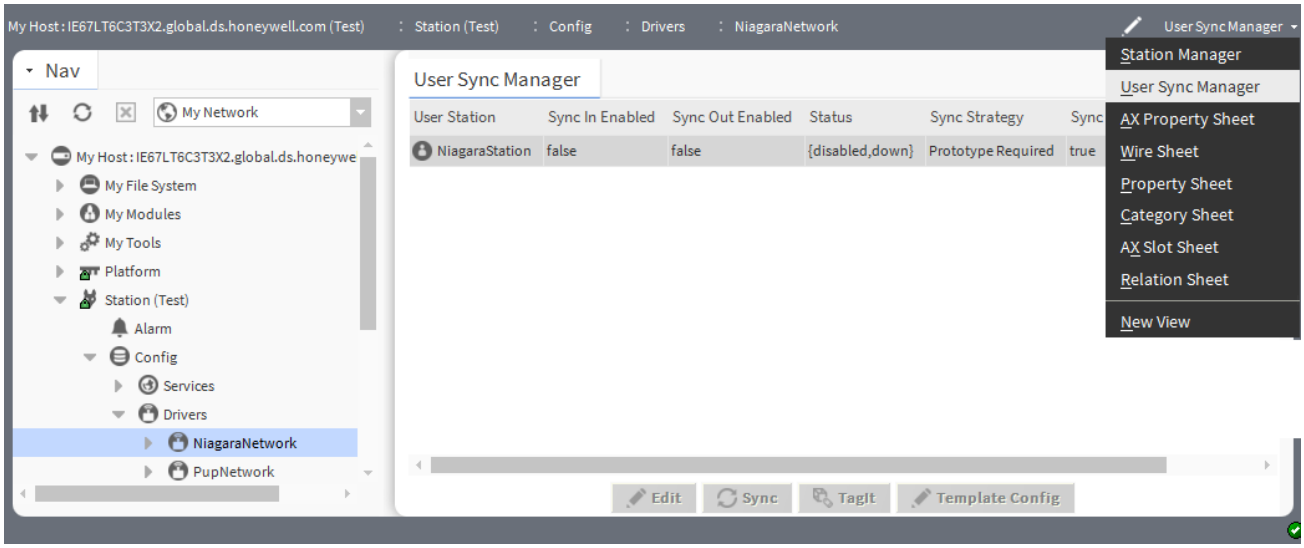
- **Template Config** accesses the station template that defines configuration options. You would select a template to set up the device with pre-configured properties.

User Sync Manager

Any station's **NiagaraNetwork** has an available **User Sync Manager** view. It manages the Users device extension properties for all **NiagaraStations** in the network.

This view reflects the **NiagaraNetwork** setup for enabling network users.

Figure 115 User Sync Manager is available view on NiagaraNetwork



To open this view, expand **Config→Drivers**, right-click the **NiagaraNetwork** and click **Views→User Sync Manager**.

For each **NiagaraStation** in the network, the **User Sync Manager** shows the current status and configuration property values of the station's user device extension. The example above, with its configuration of **`Sync In Enabled`** and **`Sync Out Enabled`** properties, represents a **NiagaraNetwork** in a Supervisor station, where all child stations are subordinate (remote controller) stations.

Columns

Column	Description
Path	Reports the ORD to the station.
User Station	Reports the name of the station.
Type	Identifies the type of station.
Sync In Enabled	Indicates if synchronization-in is available (`true`) or not (`false`).
Sync Out Enabled	Indicates if synchronization-out is available (`true`) or not (`false`).
Status	Reports the current status of the station.
Fault Cause	If the station is down or in fault, reports the cause.
Sync Strategy	Reports the value of the Synch Strategy property. `Prototype Required` or `Use Default Prototype`.
Sync Required	Indicates if synchronization is required (`true`) or not (`false`).
Sync Delay	Reports any delay time in hours, minutes and seconds (defaults to 30 seconds).

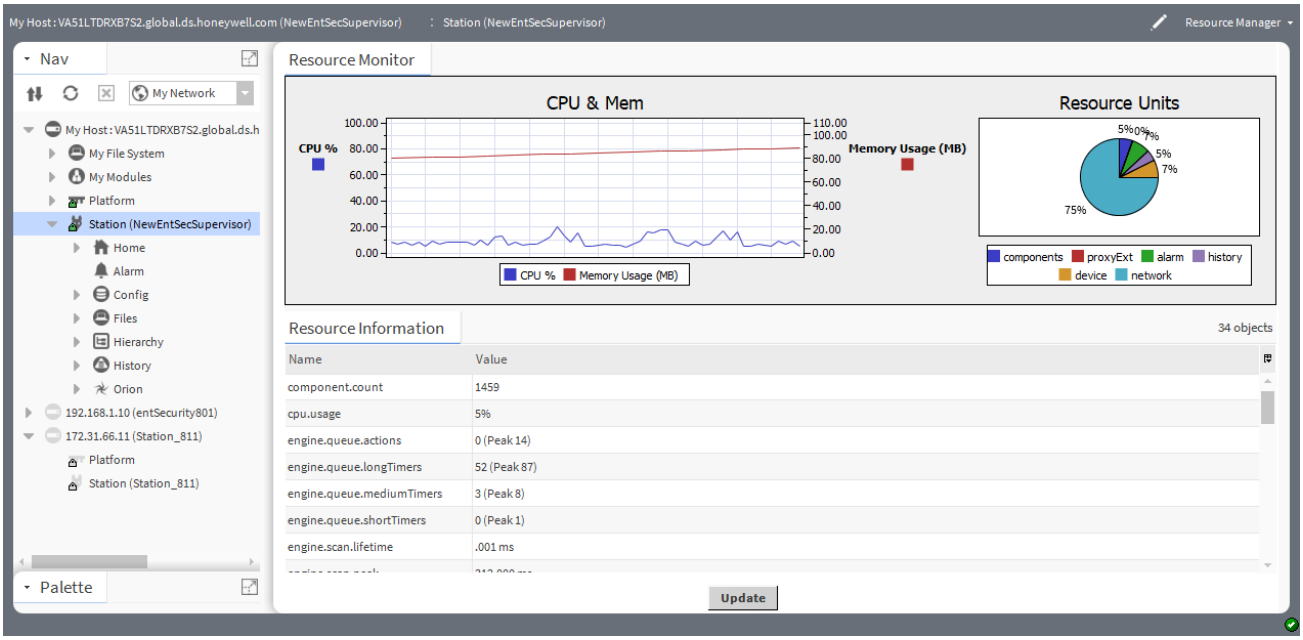
Column	Description
Sync Retry	Reports how often synchronization retries in hours, minutes and seconds (defaults to five minutes).
Last Sync Attempt	Reports the last time the framework attempted to synchronize users.
Last Sync Success	Reports the last time the framework successfully synchronized users.

Buttons

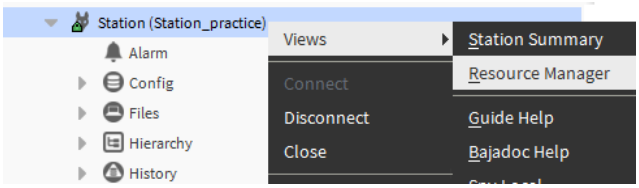
- **Edit** opens the device's database record for updating.
- **TagIt** associates metadata, such as location or unique configuration with the object.
- **Template Config** accesses the station template that defines configuration options. You would select a template to set up the device with pre-configured properties.

Resource Manager

This is a second view on the **fox-FoxSession** (**Station**) component.



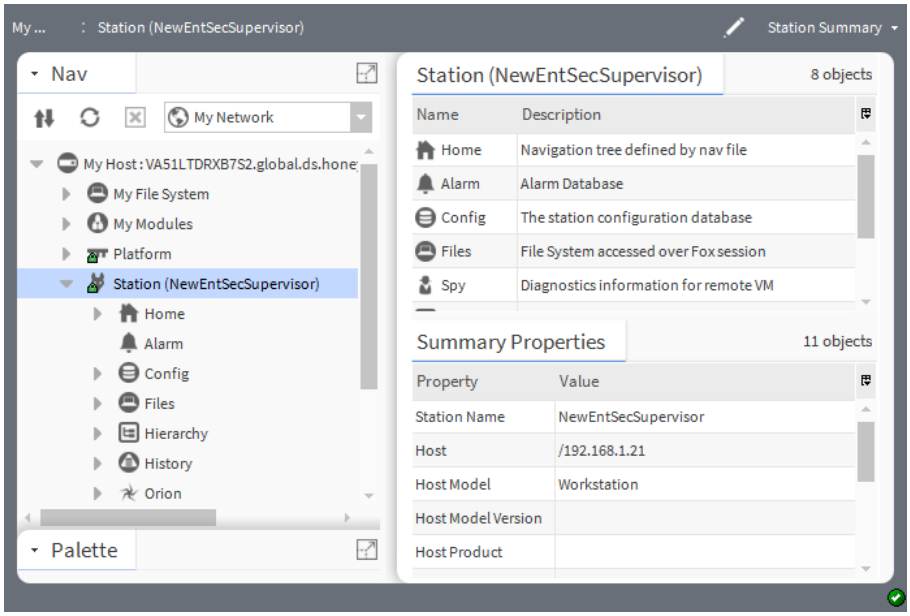
To access this view, right-click Station in the Nav tree and click **Views→Resource Manager**.



Station Summary

This is a primary view on the **fox-FoxSession** (**Station**) component.

Figure 116 Example of a Station Summary



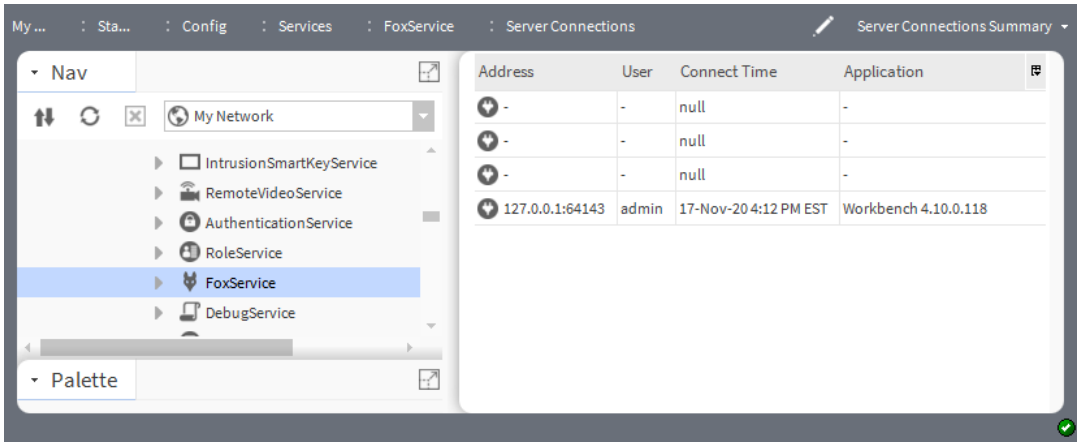
To access this view, double-click **Station** in the Nav tree.

Server Connections Summary

This plugin is the default view of the **Server Connections** slot in the **FoxService**, under in the station **Services** folder. It provides a table listing the current client connections to the station's Fox server (station-to-station connections are not included).

The main usage of this view is to perform a **Force Disconnect** action (right-click access) on any Fox server session shown. In some station maintenance scenarios, this may be helpful.

Figure 117 Server Connections Summary view



To access the **Server Connections Summary,** from the **Service Manager** expand the **Fox Service** then click the **Server Connections** slot and select **Views→Server Connections Summary**.

Following are connections summary table columns included:

Column	Description
Address	Reports the IP address of the Fox client connected to the station, along with its remote TCP port.
User	Reports the station's user account used for authentication.
Connect Time	Indicates when the Fox connection occurred.
Application	Indicates the version of Workbench being used.

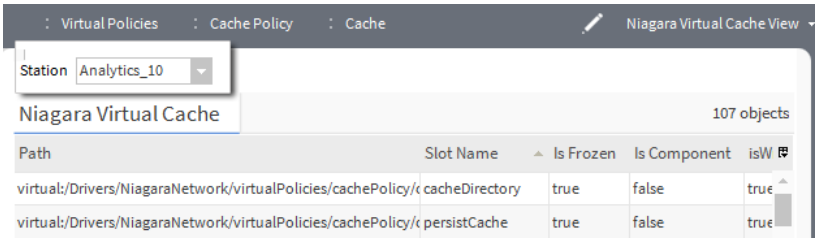
From the table, to see more details on any Fox server session, double-click an entry. The view changes to show the property sheet of that **Session**, with status slots of historical information, including connect and disconnect entries.

Niagara Virtual Cache View

Niagara Virtual Cache View is the default view on the **Cache** component child of the **Virtual Policies** container of a **NiagaraNetwork**.

Double-click the **Cache** component for its default **Niagara Virtual Cache View**.

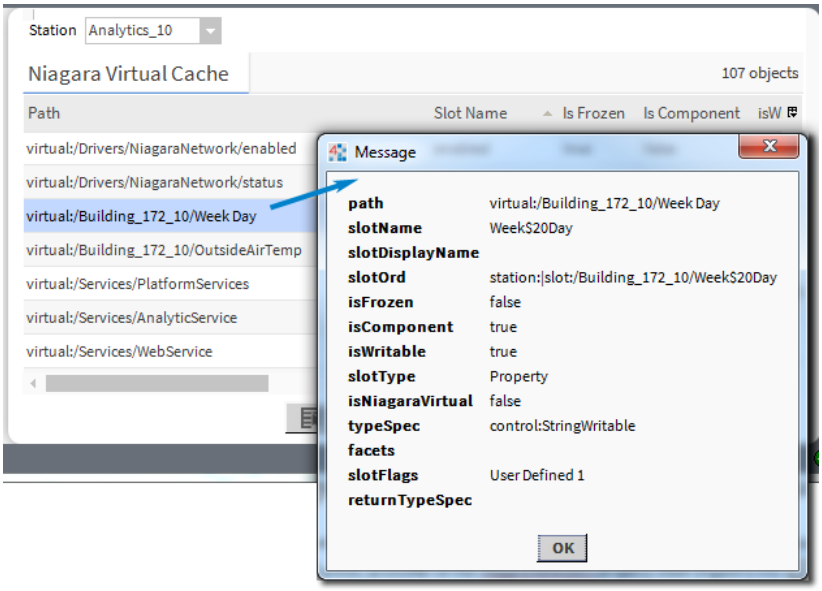
Figure 118 Niagara Virtual Cache View of the NiagaraNetwork's cache (with station select drop-down)



As shown, this tabular view lists cached virtual ords, with a station select dropdown list at the top. Click row (s) to select, where bottom buttons let you **Select All**, or **Remove** any selected.

Double-click a row for a popup window showing details on data in the virtual cache.

Figure 119 Cached details on virtual component



The cached data includes a target's slot ord along with information on its facets and slot flags.

In the case where a recent facets or slot flag change has been made to the target of a virtual component, but is not properly reflected in Px access of it, you can check it here. If necessary, you can then remove it from the virtual cache, so it can properly update and be added back in the cache upon next access.

The **Select All** is available too, which can be used NiagaraStation-wide to remove all cached data. In addition, note the Cache component has a right-click **Clear** action, accessible on the **NiagaraNetwork** property sheet.

Clear removes cached data for virtual components across all NiagaraStations in the **NiagaraNetwork**.

Nva File View

Nva File View is a view of a station's virtual archive cache file (e.g. cache1.nva) when accessed from Workbench. For related details, see "Virtual Policies (Cache) properties".

Chapter 6 Windows

Topics covered in this chapter

- New/Edit station connection
- Add station connection
- Add/Edit device
- Add/Edit point
- New/Edit file import
- New/Edit Excel Csv File Import
- New/Edit Delimited File Import
- Schedule Import New/Edit windows
- Edit schedule export window
- Edit history export window
- Add/Edit Niagara History Import
- New/Edit Niagara System History Export descriptor
- New/Edit Niagara System History Import descriptor

Windows create and edit database records or collect information when accessing a component. You access them by dragging a component from a palette to a Nav tree node or by clicking a button.

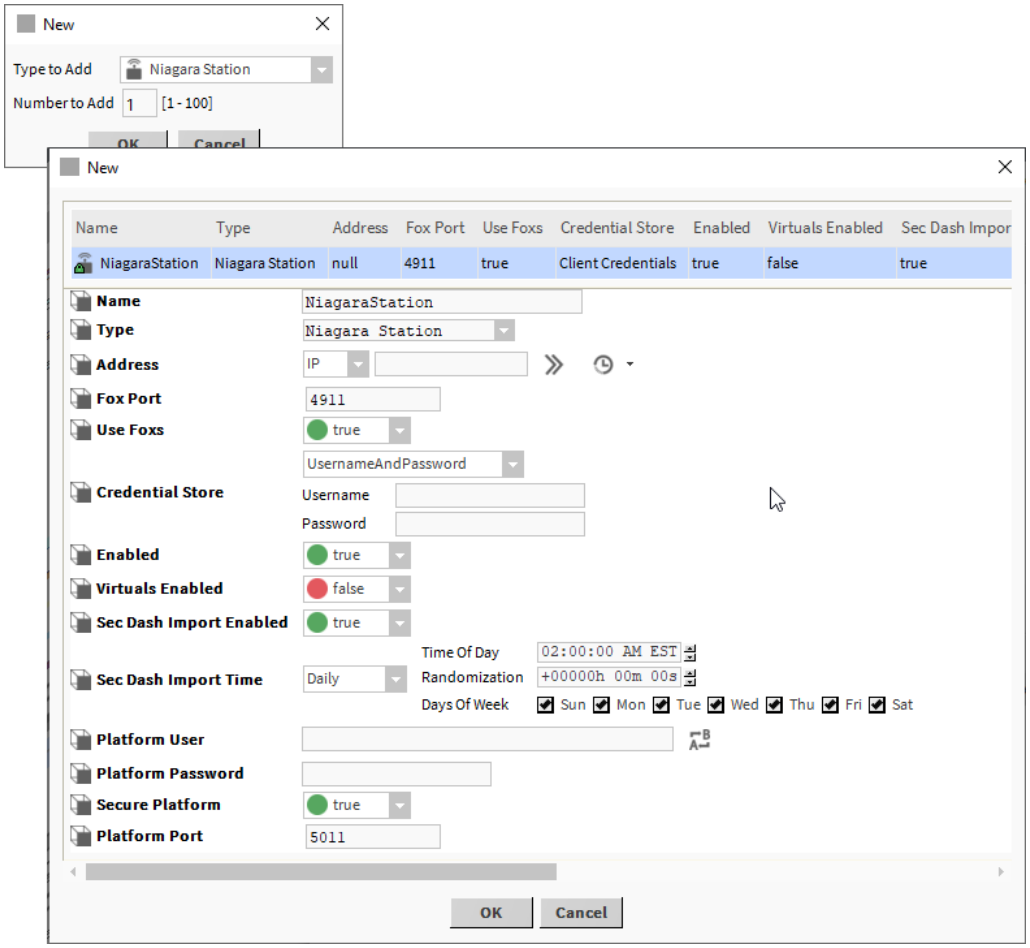
Windows do not support **On View (F1)** and **Guide on Target** help. To learn about the information each contains, search the help system for key words.

New/Edit station connection

You use the **New** windows to create a **NiagaraStation** connection and add it to the client station's database. The **Edit** window provides access to station properties for editing purposes.

Adding a **NiagaraStation** automatically creates a reciprocal, disabled server **NiagaraStation** under the **NiagaraNetwork** in the remote station.

Figure 120 Station New windows



To open this window, expand **Config→Drivers**, double-click the **NiagaraNetwork** and click **New**.

When working in a Supervisor station configured for provisioning, the **Add** (and **Edit**) windows include additional platform properties that specify the platform daemon credentials and the port, which the provisioning code uses to connect to the remote controller platform for the purpose of running provisioning jobs.

Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the configured name must match the name of the remote server station.
Type	drop-down list	Specifies the type of object. In this case, the type is a station.
Address, IP and number	additional properties	Defines the IP address of the source or destination device. In this case it identifies the IP address of the remote server station.
Fox port	Port number	Specifies the port number for secure Fox/Foxs communication to the remote device.

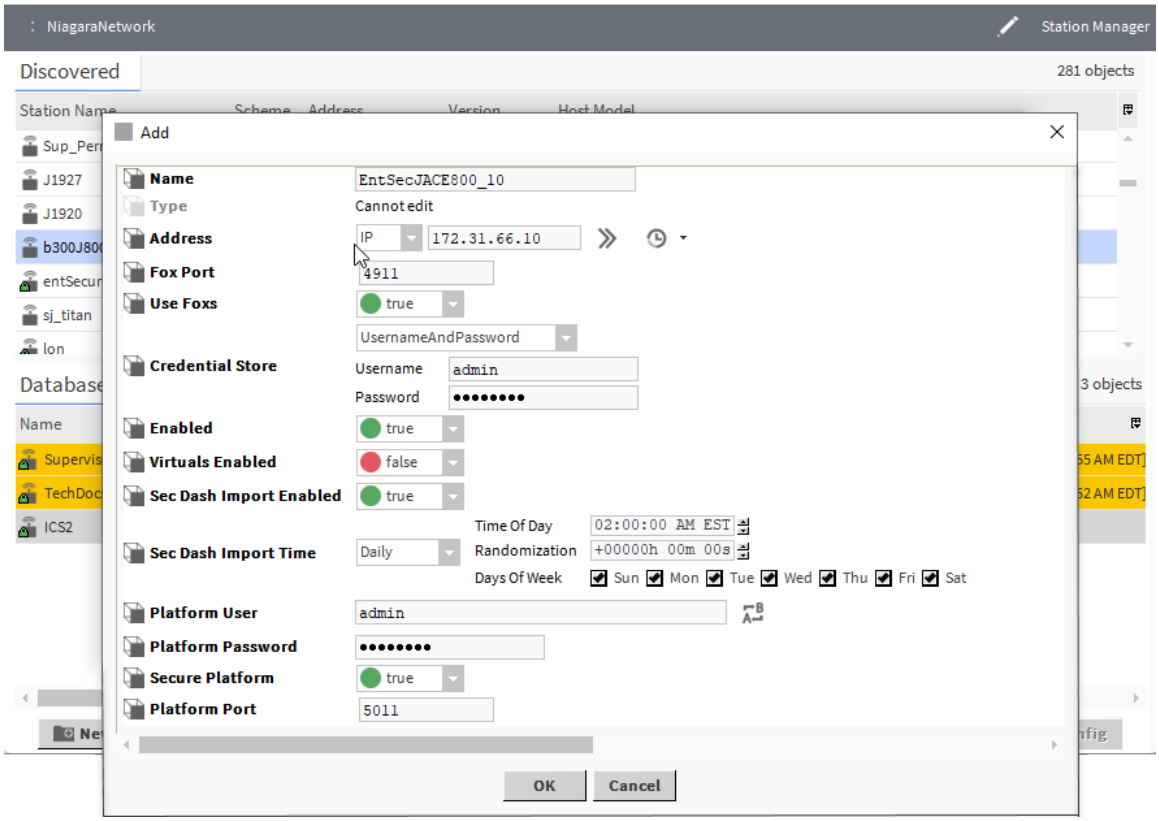
Property	Value	Description
Use Foxs	`true` (default) or `false`	Configures the station to use secure Foxs communication (`true`) or regular Fox communication (`false`).
Credential Store, Username and Password	text	Define the Username and Password required to access the device. This user name is a specific service account user previously set up in the remote station. This should be a special user for station-to-station access, with admin write privileges, and not otherwise used for normal (login) access of the remote station. This user should be unique for each project. The password should be strong. You can use a client certificate instead of a username and password for this credential store property. The *Niagara Station Security Guide* documents client certificate authentication.
Enabled	`true` (default) or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.). This property must be set to `true` for the client connection to succeed.
Virtuals Enabled	`true` or `false` (default)	Enables (`true`) and disables (`false`) access to the virtual components in this station by a user with virtual gateway permissions.
Sec Dash Import Enabled	`true` (default) or `false`	Enables (true) and disables (false) the import of the security dashboard for the **NiagaraStation**.
Sec Dash Import Time, Time of Day	hours minutes seconds (defaults to 2 am EST)	Configures when to import the security dashboard for this **NiagaraStation**.
Sec Dash Import Time, Randomization	hours minutes seconds (defaults to no set time)	Configures a random time to update the security dashboard.
Sec Dash Import Time, Days Of The Week	check boxes	Configures a specific time and day of the week to update the security dashboard.
Platform User	text	For provisioning purposes, provides the server station's platform **`Username`**. This is the name the client station sends to the remote server station to connect to it.
Platform Password	text	For provisioning purposes, provides the server station's platform **`Password`**. This is the password the client station sends to the remote server station to connect to it.
Secure Platform	`true` (default) or `false`	Enables (`true`) and disables (`false`) a secure platform connection between the client station and the remote server station for provisioning.
Platform Port	number (defaults to 5011)	Identifies the remote server platform's port.

Add station connection

This window adds a discovered station connection to the client station's **NiagaraNetwork**.

Once the client station successfully connects to the remote, server station for the first time, the system creates a reciprocal, disabled server **NiagaraStation** under the **NiagaraNetwork** of the remote station.

Figure 121 Add station window



To open this window, expand **Config→Drivers**, double-click the **NiagaraNetwork**, click **Discover**, select one or more stations and click **Add**.

When working in a client station (usually a Supervisor station) that is configured for provisioning, the last four properties in the **Add** window specify the platform daemon credentials and the port for the selected platform. The provisioning code uses this information to connect to the remote controller platform for the purpose of running provisioning jobs.

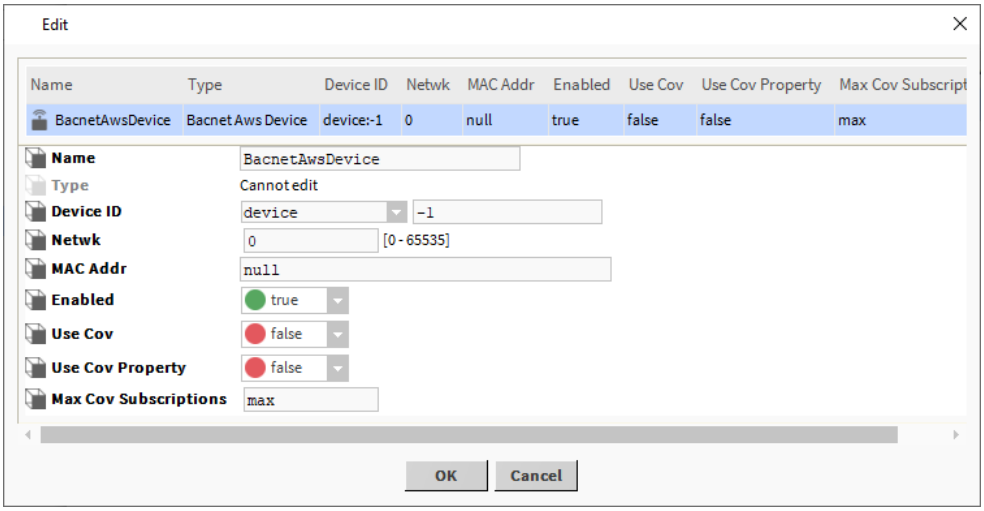
Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is a server station.
Type	drop-down list	Specifies the type of object. In this case, the type is a station.
Address	additional properties	Defines the IP address of the source or destination device. In this case it identifies the station on the network.

Property	Value	Description
Foxs port	Port number	Specifies the port number for secure Fox/Foxs communication to the remote device.
Use Foxs	`true` (default) or `false`	Configures the station to use secure Foxs communication (`true`) or regular Fox communication (`false`).
Credential Store, Username and Password	text	Define the Username and Password required to access the device. This user name is a specific service account user previously set up in the remote station. This should be a special user for station-to-station access, with admin write privileges, and not otherwise used for normal (login) access of the remote station. This user should be unique for each project. The password should be strong.
Enabled	`true` (default) or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Virtuals Enabled	`true` or `false` (default).	Determines whether virtual components can be accessed in this station, by any user with permissions on its Virtual Gateway.

Add/Edit device

These windows add and edit device records. This topic documentsonly some of a device component's properties. It uses the BACnetAws device window as an example of the type of information required to configure a device.

Figure 122 Example of a BACnet Aws device Edit window (single device)



To open this type of window, expand **Config→Drivers**, double-click the network node in the Nav tree, click **Discover**, select a one or more discovered device(s) and click **Add**.

The **Add** window is nearly identical to the device **Edit** window, but allows you to edit **Type** as well as other device properties. Often, device address properties in the **Add** window already have acceptable values for operation (otherwise, communications to that device would not have occurred). Usually, you change only

Name unless you know of other properties that need to change. You can always edit the device component (s) after you click **OK** and add them to your station database.

NOTE: When one or more devices are selected in the top **Discovered** pane, the Create New Objects tool (icon) is also available on the toolbar ( ), as well as the **New** and **Edit** commands in the **Manager** menu. Also, you can simply double-click a discovered device to open its **Edit** window.

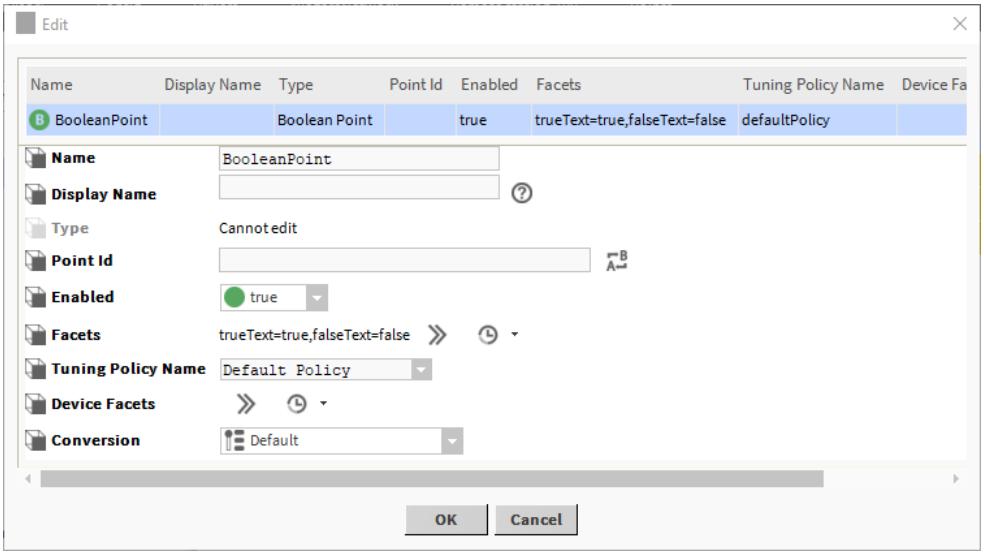
Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is a device.
Type	drop-down list	Specifies the type of object. In this case, the type is a device. When a single device is selected in the **Edit** window, you can edit any property except **Type** because its value was fixed when you created the new device.
Device Id, list and instance	drop-down list and text	Provide a name for the device. The two values together ensure a unique name. These values do not change the device's ID in the station database.
Netwk	network	This is exclusively a BACnet feature. For more information, refer to the *Niagara 4 BACnet Driver Guide*
Mac Addr	text	Specifies the data link layer MAC address of the device.
Enabled	`true` (default) `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).
Use Cov	`true` or `false` (default)	Enables (`true`) and disables (`false`) a device's support for COV (change of value) as a way to monitor proxy point values. This is exclusively a BACnet feature. For more information, refer to the *Niagara 4 BACnet Driver Guide*
Use Cov Property	`true` or `false` (default)	Configures a tuning policy to process COV notifications. This is exclusively a BACnet feature. For more information, refer to the *Niagara 4 BACnet Driver Guide*
Max Cov Subscription	Text (defaults to `max`)	Specifies the maximum number of COV (change of value) subscriptions that the database attempts to use with this device. This is exclusively a BACnet feature. For more information, refer to the *Niagara 4 BACnet Driver Guide*

Add/Edit point

This type of window adds and edits proxy points in the Supervisor station's database. **Edit** does not apply to point folders.

This window configures mostly proxy extension properties plus the parent point's **Name** and **Device Facets**. These properties vary among drivers. To access all properties of the proxy point, including all those under any of its extensions, go to its **Property Sheet**.

Figure 123 Edit point properties



To open an instance of this window under the **NiagaraNetwork**, expand **Config→Drivers→NiagaraNetwork**, expand a **NiagaraStation**, double-click the **Points** folder and either click **New** or select an existing point and click **Edit**.

To open this type of window under other driver networks, expand **Config→Drivers**, expand the network (for example **BacnetNetwork**), double-click its **Points** folder and either click **New** or select an existing point and click **Edit**.

You can select one or more proxy points in a station to edit.

Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is a point.
Display Name	text	Defines a BFormat string used to format text by using values obtained from objects. You specify this string as normal text with embedded scripts identified by the percent (%) character. The driver maps calls within the script to an object's methods. Use the dot operator (.) to chain calls. To insert a percent symbol itself, use two percent symbols (%%). For examples, click the question mark icon next to this property.
Type	drop-down list	Specifies the type of object. In this case, the type is a point.
Point Id	text	Specifies an ID for the point.
Enabled	`true` (default) or `false`	Activates (`true`) and deactivates (`false`) use of the object (network, device, point, component, table, schedule, descriptor, etc.).

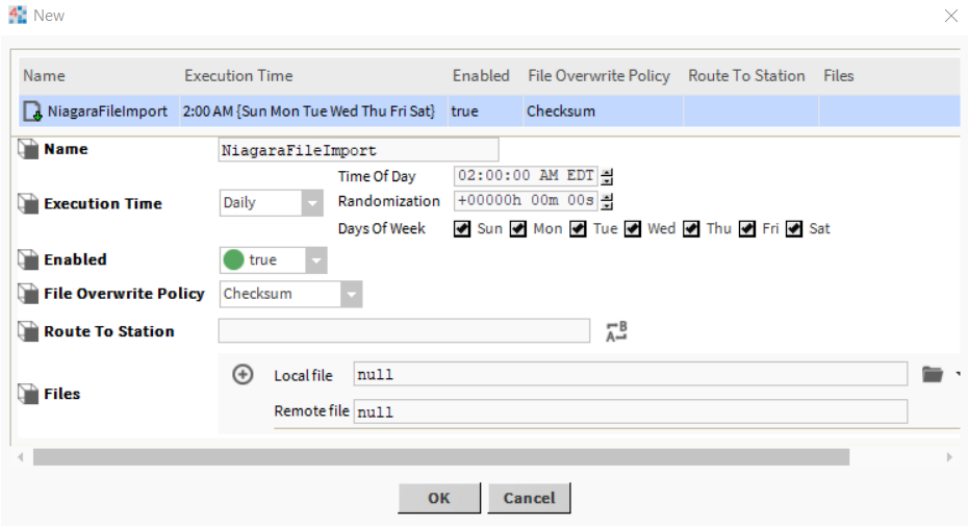
Property	Value	Description
Facets	Config Facets window	Sets up the text string that represents true and false values. For example, instead of `true` and `false`, the strings could be `yes` and `no` or `enabled` and `disabled`.
Tuning Policy Name	drop-down list	Selects a network tuning policy by name. This policy defines stale time and minimum and maximum update times. During polling, the system uses the tuning policy to evaluate both write requests and the acceptability (freshness) of read requests.

Property	Value	Description
Device Facets	Config Facets window	Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types. With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value. Besides control points, various other components have facets too. For example, many **`kitControl`** and schedule components have facets. Details about point facets apply to these components too, unless especially noted. You access facets by clicking an **Edit** button or a chevron **>>**. Both open an **Edit Facets** window.
Conversion	drop-down list	Defines how the system converts proxy extension units to parent point units. `Default` automatically converts similar units (such as Fahrenheit to Celsius) within the proxy point. **NOTE:** In most cases, the standard `Default` conversion is best. `Linear` applies to voltage input, resistive input and voltage output writable points. Works with linear-acting devices. You use the Scale and Offset properties to convert the output value to a unit other than that defined by device facets. `Linear With Unit` is an extension to the existing linear conversion property. This specifies whether the unit conversion should occur on "Device Value" or "Proxy Value". The new linear with unit convertor, will have a property to indicate whether the unit conversion should take place before or after the scale/offset conversion. `Reverse Polarity` applies only to Boolean input and relay output writable points. Reverses the logic of the hardware binary input or output. `500 Ohm Shunt` applies to voltage input points only. It reads a 4-to-20mA sensor, where the Ui input requires a 500 ohm resistor wired across (shunting) the input terminals. `Tabular Thermistor` applies to only a Thermistor input point and involves a custom resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors. `Thermistor Type 3` applies to an Thermistor Input point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors. `Generic Tabular` applies to 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.

New/Edit file import

This window configures the import descriptor used to import files from a remote controller station to a Supervisor station. File import is a function of the **niagaraDriver** module.

Figure 124 New window for Niagara File Import



To open this window add a **Files** component from the **niagaraDriver** palette to the **Files** node in the station, double-click the added node and click **New**. To edit an existing file import record, click **Edit**.

In addition to the standard property, Enabled these properties configure this window.

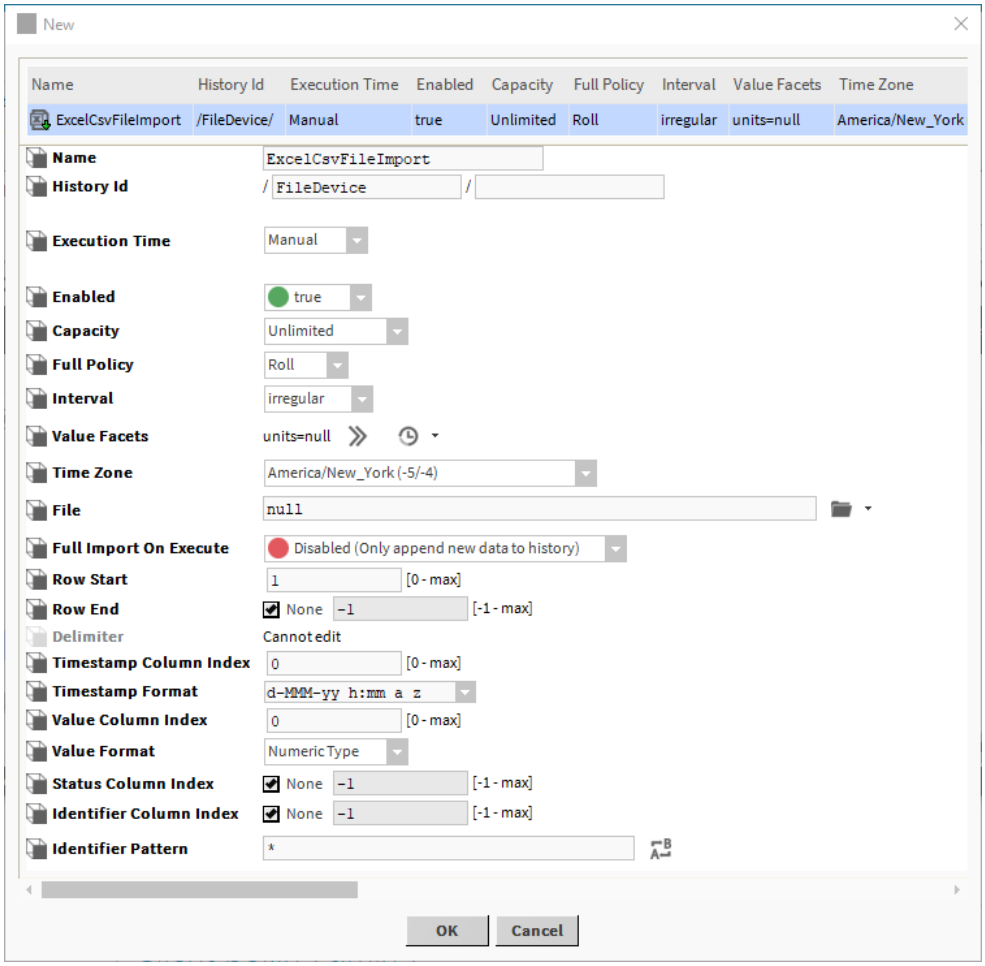
Property	Value	Description
Name	text (defaults to `NiagaraFileImport`)	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component.In this case, the object is a file import descriptor. You can leave this value at the default and append numbers as needed to keep the name unique. Editing this name does not affect the names of imported file(s).
Execution Time	drop-down list with additional properties (defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Execution Time, Time Of Day	hours minutes seconds (defaults to 2 am)	Configures when, during the day, to import files. defines start and end times for the interval. The default is 24–hours, that is, start at 12:00 am and end at 11:59 pm.
Execution Time, Randomization	hours minutes seconds (defaults to zero, that is no randomization)	Calculates the next execution time and adds a random amount of time between zero milliseconds to the calculated time. This option may prevent server flood issues if too many file imports are executed at the same time. Default is zero (no randomization).
Execution Time, Days of the Week	check boxes (defaults to all days)	Specifies the days of the week to include in the function.

Property	Value	Description
File Overwrite Policy	drop-down list (defaults to `Checksum`)	Selects the criterion to apply to overwrite existing files upon any import execution: `Checksum` compares a checksum from the file being imported with the checksum from the existing file. If the app finds a difference, it imports the file again and overwrites the existing file. `Last Modified` compares the date and time stamps of the existing and imported files. Any remote files found with a more recent date or time overwrite the existing files.
Route To Station	text	As of Niagara 4.13 and later, optionally specifies a route of intermediate station names (delimited by semi-colons) to reach the remote, reachable station from which to perform the file import operation. If the field remains empty, the file is imported from the directly connected station (just like the pre-Niagara 4.13 behavior). If a station route is provided, the file is imported from the downstream reachable station (the final station name in the list) and transmitted through all intermediate stations, as indicated by the station route, back up to this station. In order to work, it requires active NiagaraNetwork connections for all intermediate stations specified in the route.
Files	File Selector	Selects the receiving target and sending source file (or directory) pair(s).In this case, the receiving file (or directory) is in the local system and the sending file (or directory is in the remote system.
Files, Local File	file path	Defines the location of the local file in the local station's file space. If you used a discovery job to add, this value matches the initially selected remote file value. If the local directories in the path do not exist, the station creates them upon import execution.
Files, Remote File	file path	Defines the location of the file to be imported from the remote station's file space. This reflects the selected directory or file for the add job. Typically, you do not edit this value.

New/Edit Excel Csv File Import

This window sets up a CSV file that contains history data for import to, usually, a Supervisor station. CSV file import is a function of the **`driver`** module.

Figure 125 New Excel Csv File Import properties



To open this window, expand **Config→Drivers→FileNetwork→FileDevice→Histories**, double-click an archive folder and click the **New** button.

Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component.In this case, the object is a CSV file import descriptor.
History Id	text in two parts: station name/history name	Together the two names identify the CSV file to import from the local Supervisor PC.
Execution Time	drop-down list (defaults to `Daily`)	Selects when to import or export descriptors that a join creates automatically on demand. `Daily` opens additional properties to configure time of day and day of the week. `Manual` configures execution to take place on demand. `Interval` configures execution to take place on a regular basis.

Property	Value	Description
Capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Full Policy	drop-down list (defaults to `Roll`)	Defines what should happen if **`Capacity`** is set to `Record Count` and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.
Interval	drop-down list (defaults to `irregular`)	Sets up an automatic function, such as the import or export of data, by specifying how often to execute the function.
Value Facets	drop-down list (defaults to `Numeric Type`)	Identifies the type of data contained in the file.
Time Zone	drop-down list (defaults to `America/New_York (-5/-4)`)	Selects the suitable time zone from the drop-down list.
File	file path	Selects the path to the CSV file.
Full Import On Execute	drop-down list	Configures how much data to import. **`Enabled`** transfers the entire history on each import. **`Disabled`** appends only new data to the history on each import.
Row Start	number	Specifies the line number in the file from which to begin importing data.
Row End	number	Specifies the line number in the file at which to stop importing data.
Delimiter	n/a	Unavailable for edit.
Timestamp Column Index	number	Specifies a left-to-right index (zero-based) to identify the column in the file from which to import the timestamp.
Timestamp Format	drop-down list (defaults to `d-MMM-yy h:mm a z`)	Selects the timestamp format from the drop-down list.
Value Column Index	number (defaults to zero (0))	Identifies the column in the record that contains the value.
Value Format	drop-down list (defaults to `Numeric Type`)	Selects the value format from the drop-down list.

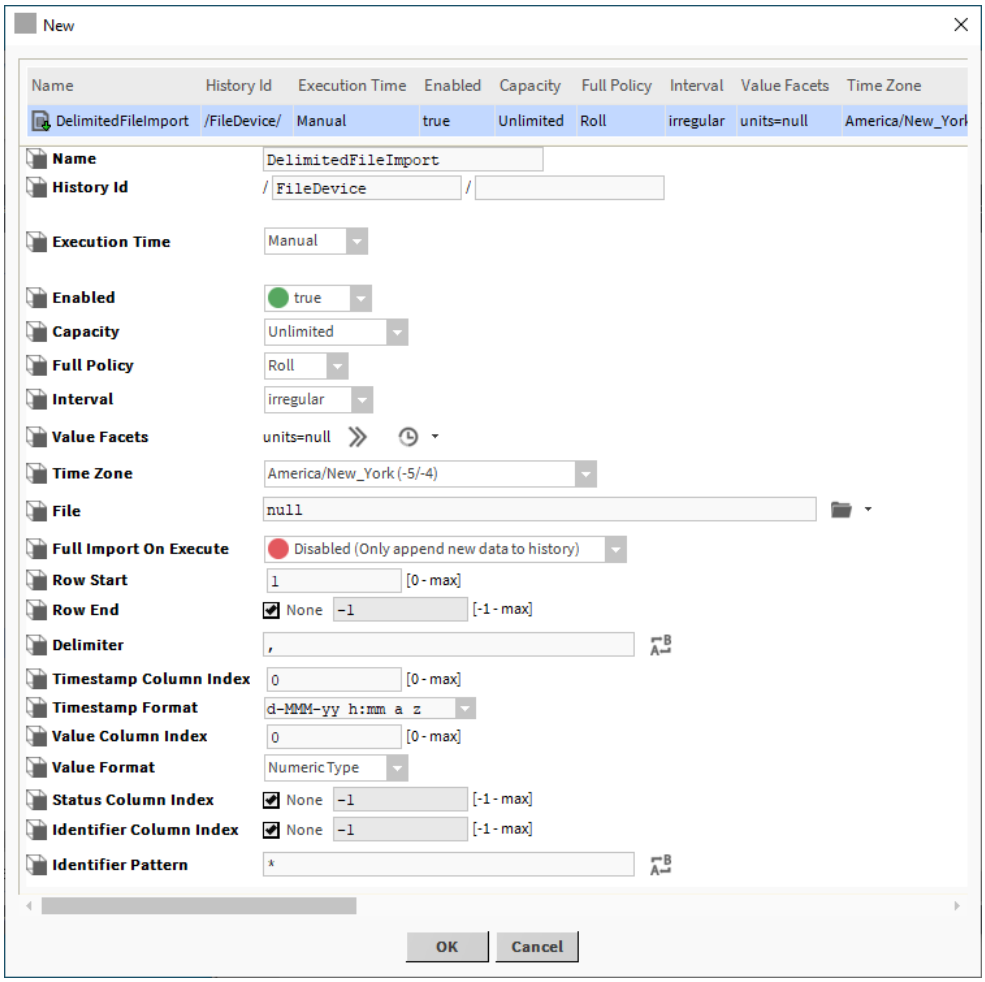
Property	Value	Description
Status Column Index	number	Specifies the number for the left-to-right index (zero-based) to identify the column in the file used to import the status, if available.
Identifier Column Index	number	Specifies the number for the left-to-right index (zero-based) to identify the column in the file used to filter rows for inclusion in combination with the **`Identifier Pattern`** property value).
Identifier Pattern	text (defaults to asterisk (*))	Specifies the text string that import searches for in all rows, where the import job imports any row with the matching text string.

New/Edit Delimited File Import

This window defines the contents of a comma-delimited or tab-delimited external file to import into a Supervisor station's **FileNetwork**, a component in the **`driver`** palette.

This window requires a **FileNetwork** component in the **Drivers** container, with a **FileDevice**, which includes A file history device extension.

Figure 126 Delimited File Import properties



To open this window, expand **Config→Drivers→FileNetwork→FileDevice**, double-click **Histories**, click **New**, select `Delimited File Import` from the drop-down list and click **OK**.

In addition to the standard property, Enabled, these properties are available in this window.

Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is a delimited file import descriptor.
History Id	text in two parts: station name/history name	Together the two names identify the CSV file to import from the local Supervisor PC.
Execution Time	drop-down list (defaults to `Daily`)	Selects when to import or export descriptors that a join creates automatically on demand. `Daily` opens additional properties to configure time of day and day of the week. `Manual` configures execution to take place on demand. `Interval` configures execution to take place on a regular basis.
Capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Full Policy	drop-down list (defaults to `Roll`)	Defines what should happen if **`Capacity`** is set to `Record Count` and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.
Interval	drop-down list (defaults to `irregular`)	Sets up an automatic function, such as the import or export of data, by specifying how often to execute the function.
Value Facets	drop-down list (defaults to `Numeric Type`)	Identifies the type of data contained in the file.
Time Zone	drop-down list (defaults to `America/New_York (-5/-4)`)	Selects the suitable time zone from the drop-down list.
File	file path	Selects the path to the CSV file.
Full Import On Execute	drop-down list	Configures how much data to import. **`Enabled`** transfers the entire history on each import.

Property	Value	Description
		Disabled appends only new data to the history on each import.
Row Start	number	Specifies the line number in the file from which to begin importing data.
Row End	number	Specifies the line number in the file at which to stop importing data.
Delimiter	n/a	Unavailable for edit.
Timestamp Column Index	number	Specifies a left-to-right index (zero-based) to identify the column in the file from which to import the timestamp.
Timestamp Format	drop-down list (defaults to `d-MMM-yy h:mm a z`)	Selects the timestamp format from the drop-down list.
Value Column Index	number (defaults to zero (0))	Identifies the column in the record that contains the value.
Value Format	drop-down list (defaults to `Numeric Type`)	Selects the value format from the drop-down list.
Status Column Index	number	Specifies the number for the left-to-right index (zero-based) to identify the column in the file used to import the status, if available.
Identifier Column Index	number	Specifies the number for the left-to-right index (zero-based) to identify the column in the file used to filter rows for inclusion in combination with the **Identifier Pattern** property value).
Identifier Pattern	text (defaults to asterisk (*))	Specifies the text string that import searches for in all rows, where the import job imports any row with the matching text string.

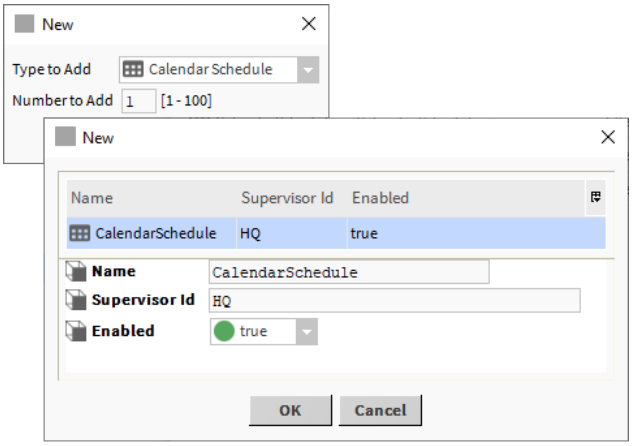
Schedule Import New/Edit windows

In the **Schedule Import Manager**, the **New** and **Edit** windows configure schedule import extension properties. This is a **NiagaraNetwork** view.

The **New** button in a device's **Schedule Import Manager** is used only if engineering offline (all devices with a Schedules extension support online discovery). If used, **Match** may be used later (when online with the device).

A **New** tool is also on the **Schedule Import Manager** toolbar, and in the **Manager** menu.

Figure 127 New window for schedule import properties



To open these windows, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, double-click **Schedules** and click **New**.

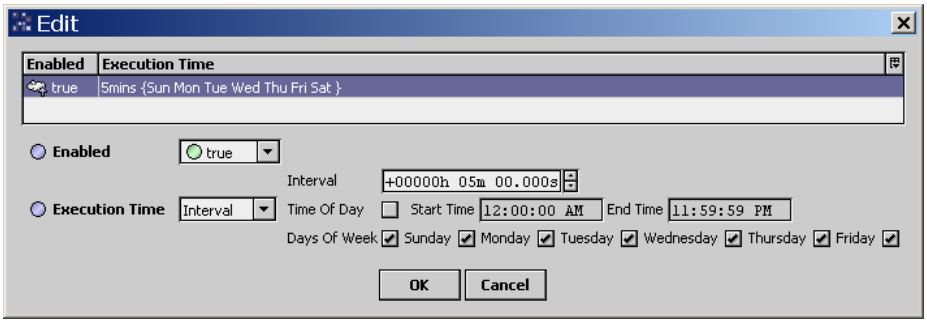
In addition to the standard property, Enabled, these properties support a schedule import.

Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is an imported schedule component. If discovered, this text matches the name of the source schedule. This must be a unique name among the other components in same container. **NOTE:** Editing this name does not affect the name of the source schedule nor the name of the corresponding schedule export descriptor.
Supervisor Id(**NiagaraStation** only)	text	Identifies the Supervisor station.

Edit schedule export window

This window configures schedule export descriptor properties.

Figure 128 Edit schedule export properties



In addition to the standard property, Enabled, this window provides some properties from the control time trigger.

Property	Value	Description
Execution Time	drop-down list (defaults to `Interval`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide. In this case, the function makes an export update to the remote (imported) schedule, providing that a configuration change occurred in the local schedule that requires synchronization (export).
Execution Time, Interval	hours minuets seconds	Sets up an automatic function, such as the import or export of data, by specifying how often to execute the function.
Execution Time, Time of Day	start and end hour	Configures when, during the day, to import files. defines start and end times for the interval. The default is 24–hours, that is, start at 12:00 am and end at 11:59 pm.
Execution Time, Days of the Week	check boxes (defaults to daily)	Specifies the days of the week to include in the function.

Edit history export window

The **Edit** window shows the configuration properties of the history export descriptor, plus **Name**, which is equivalent to the right-click **Rename** command on the descriptor. To access all properties, including all status properties, view the HistoryService property sheet.

The **NiagaraHistoryExport** line above the properties summarizes the properties.

In addition to the standard property, Enabled, these properties support history export.

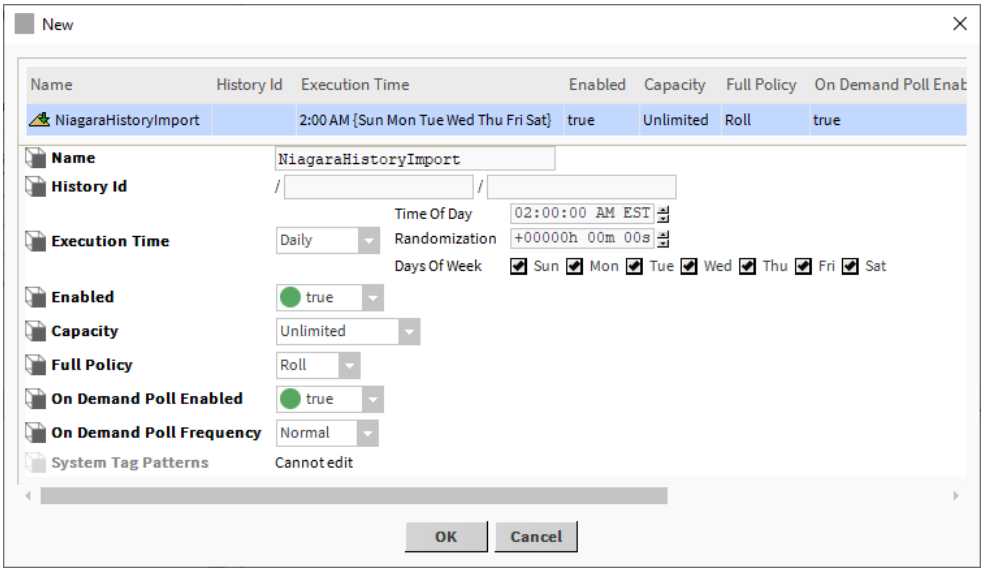
Property	Value	Description
Name	text and numbers	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. For a history originating in the local host station, this name begins with `Local_` . For a system history export, originating in the remote station, the name begins with `NiagaraSystemHistoryExport`. If discovered for import, you typically leave this name at its default.
Execution Time	drop-down list (defaults to `Manual`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide. In this case, the function makes an export update to the remote (imported) history, providing that a configuration change occurred in the local history and requires synchronization (export).
HistoryId	text in two parts: station name/history name	Together the two names identify the CSV file to import from the local Supervisor PC.

Property	Value	Description
Execution Time, Time of Day	start and end hour	Configures when, during the day, to import files. defines start and end times for the interval. The default is 24–hours, that is, start at 12:00 am and end at 11:59 pm.
Execution Time, Interval	hours minuets seconds	Sets up an automatic function, such as the import or export of data, by specifying how often to execute the function.

Add/Edit Niagara History Import

This window configures a history import descriptor.

Figure 129 Niagara History Import properties



To open this window, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, double-click **Histories**, click **New** or select a descriptor and click **Edit**.

In addition to the standard property, Enabled, these properties support import descriptors.

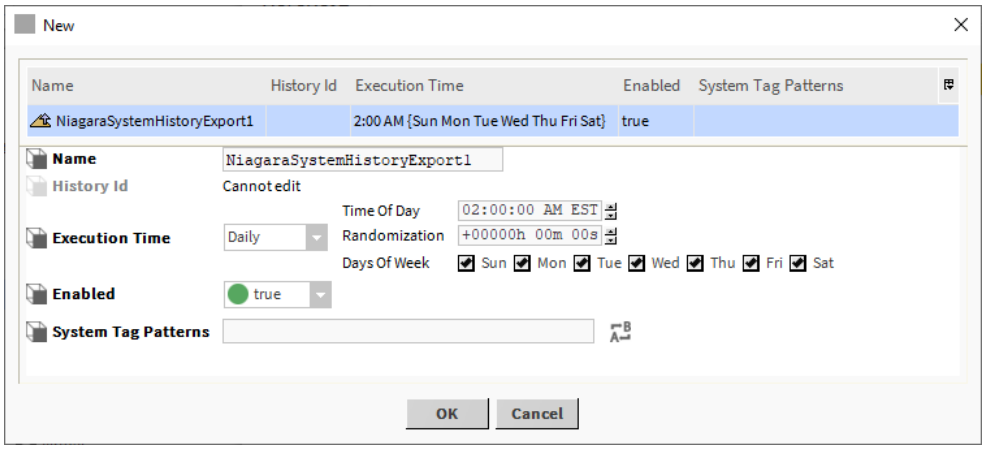
Property	Value	Description
Name	text	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component. In this case, the object is a history import descriptor.
History Id	text in two parts: station name/history name	Together the two names identify the CSV file to import from the local Supervisor PC.
Execution Time	drop-down list (defaults to `Daily`)	Selects when to import or export descriptors that a join creates automatically on demand. `Daily` opens additional properties to configure time of day and day of the week. `Manual` configures execution to take place on demand.

Property	Value	Description
		`Interval` configures execution to take place on a regular basis.
Capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records. `Record Count` opens an additional property for defining the table limit.
Full Policy	drop-down list (defaults to `Roll`)	Defines what should happen if **`Capacity`** is set to `Record Count` and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.
On Demand Poll Enabled	`true` (default) or `false`	Enables and disables polling. `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
On Demand Poll Frequency	drop-down list	References the **`On Demand Poll Scheduler`** rates under the **NiagaraNetwork**'s **History Policies** container slot.
System tags	n/a	Cannot be edited.

New/Edit Niagara System History Export descriptor

This window sets up a history export descriptor that uses tags instead of history IDs to export histories.

Figure 130 System History Export properties



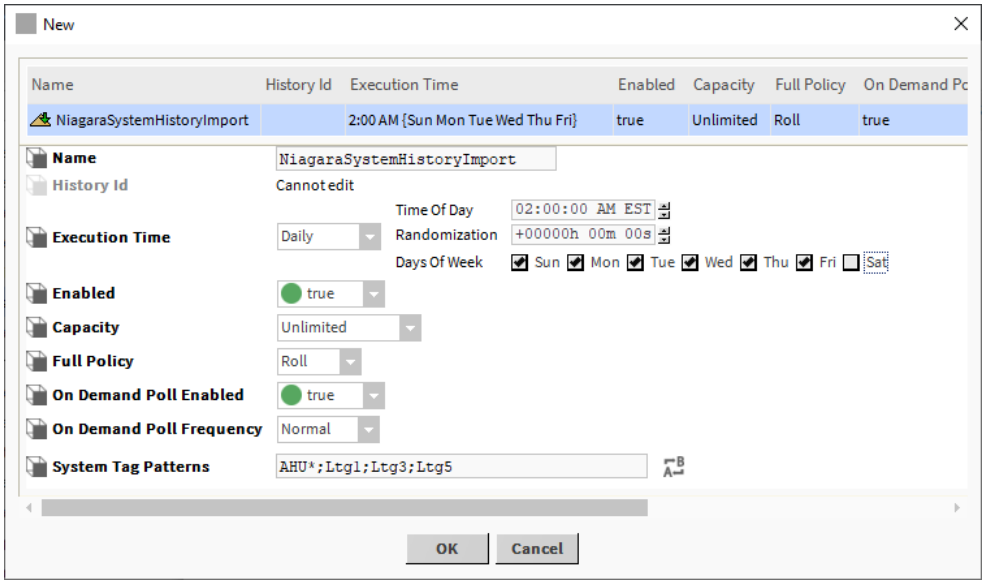
To open this window, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, double-click **Histories** click **New**, select `Niagara System History Export` from the drop-down **`Type to Add`** menu and click **OK**.

Property	Value	Description
Name	text (default begins with `Local_` for any history originating from the local station. If you are adding a descriptor this property defaults to `NiagaraSystemHistoryExport` appending numerals as needed to keep unique.)	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component.In this case, the object is a history export descriptor. Typically you leave it at its default value. Editing this name does not affect the name of resulting history that was already exported.
History Id	text in two parts: station name/history name	Together the two names identify the CSV file to import from the local Supervisor PC. Histories originating in the local station show a caret (^), which is shorthand for the local station, and source history names. Typically, you leave both at their default values.
Execution Time	additional properties (**`Trigger Mode`** defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
System Tag Patterns		Specifies one or more text strings matched against text values in the System Tags properties of a local history extensions, where matching text patterns result in histories exported into the remote history space.

New/Edit Niagara System History Import descriptor

This window sets up a history import descriptor that uses tags instead of history IDs to import histories.

Figure 131 System History Import properties



To open this window, expand **Config→Drivers→NiagaraNetwork→NiagaraStation**, double-click **Histories** click **New**, select `Niagara System History Import` from the drop-down **`Type to Add`** menu and click **OK**.

Property	Value	Description
Name	text (default begins with `Local_` for any history originating from the local station. If you are adding a descriptor this property defaults to `NiagaraSystemHistoryImport` appending numerals as needed to keep unique.)	Provides a name for the object. Editing this property is equivalent to the right-clicking **Rename** command on the component.In this case, the object is a system history import descriptor . Typically you leave this name at its default value. Editing this name does not affect name of resulting history that was already exported into a remote station.
History Id	text	Together the two names identify the CSV file to import from the local Supervisor PC. Histories originating in the local station show a caret (^), which is shorthand for the local station, and source history names. Typically, you leave both at their default values.
Execution Time	additional properties (**`Trigger Mode`** defaults to `Daily`)	Configures a time trigger that controls when to perform the function. Trigger properties are documented in the *Getting Started with Niagara* guide.
Capacity	drop-down list (defaults to `Unlimited`)	Defines the maximum number of history records allowed in the associated table. `Unlimited` enforces no limitation on the number of records.

Property	Value	Description
		`Record Count` opens an additional property for defining the table limit.
Full Policy	drop-down list (defaults to `Roll`)	Defines what should happen if **`Capacity`** is set to `Record Count` and the specified record count is reached. `Roll` overwrites the oldest records with the newest ones. This ensures that the latest data are recorded. `Stop` terminates recording when the number of stored records reaches the specified capacity. Full policy has no effect if **`Capacity`** is `Unlimited`.
On Demand Poll Enabled	`true` (default) or `false`	Enables and disables polling. `true` enables a system user to use the **Live Updates** (play) button in History views to poll for live data for the associated imported history(ies). `false` renders this button unavailable in history views for the associated imported history(ies).
On Demand Poll Frequency	drop-down list	References the **`On Demand Poll Scheduler`** rates under the **NiagaraNetwork**'s **History Policies** container slot.
System Tag Patterns	text	Specifies one or more text strings matched against text values in the System Tags properties of a local history extensions, where matching text patterns result in histories exported into the remote history space.

Index

M

N

O

P

R

S

T

U

V

W

Glossary

device	With regard to the Niagara network, a device is an object that fulfills a particular purpose. A device may be a physical mechanical or electrical component, such as a BACnet device, or it may be a software object, such as a station, oBIX server. What actually comprises a device may depend on your design and levels of abstraction. Devices can provide or accept different types of data and they may contain metadata, such as device ID, Model, or serial number of the device itself.
DIBS	Distributed Internet Backup System
job	The mechanism used to manage a task that a station performs. Jobs run asynchronously in the background while providing user visibility regarding what is going on in the station. See also, provisioning job.
ORD	An ORD is an "Object Resolution Descriptor". The ORD is the Niagara universal identification system and is used throughout the framework. The ORD unifies and standardizes access to all information. It is designed to combine different naming systems into a single string and has the advantage of being parsable by a host of public APIs.
worker	A programming term for a system process or thread, which is the smallest sequence of programmed instructions that an operating system can manage independently using a scheduler.