Technical Document

NiagaraAX Drivers Guide

AX-3.8 and AX-3.7u1

November 5, 2013

Niagara^{AX} Drivers Guide

Confidentiality Notice

Trademark Notice

Copyright and Patent Notice

Contents

Preface . . . vii

Document Change Log . . . vii

Driver architecture . . . 1–1

What are networks? . . . 1–1
- Where are networks located? . . . 1–1

About Network architecture . . . 1–2
- Network component hierarchy . . . 1–2
- About network components . . . 1–3
- About device components . . . 1–3
- About device extensions . . . 1–4

About the Driver Manager . . . 1–4
- Driver Manager New and Edit . . . 1–4

Common network components . . . 1–5
- Network status properties . . . 1–6
- About network Alarm Source Info . . . 1–6
- About Monitor . . . 1–7
- About Tuning Policies . . . 1–8
- About poll components . . . 1–10
- Using poll statistics in tuning poll rates . . . 1–13
- Additional network components . . . 1–13

About the Device Manager . . . 1–14
- Device New Folder and New . . . 1–15
- All Descendants . . . 1–16
- Device Edit . . . 1–17
- About Device Discover, Add and Match (Learn Process) . . . 1–18
- Match (Device) . . . 1–20
- Manager table features . . . 1–21

Common device components . . . 1–21
- Device status properties . . . 1–22
- Device Alarm Source Info . . . 1–22
- Device address properties . . . 1–23
- Driver-specific device slots . . . 1–23

Virtual gateway and components . . . 1–23
- About virtual component spaces . . . 1–24
- About virtual gateways . . . 1–24
- Application and limitations of virtual components . . . 1–25
- Virtual components in Px views . . . 1–25
- Virtual ord syntax . . . 1–26

Types of device extensions . . . 1–26

About the Points extension . . . 1–27
- About proxy points . . . 1–27
- ProxyExt properties . . . 1–32
- Proxy point status . . . 1–33
- Effect of facets on proxy points . . . 1–33

About the Histories extension 1–34
About the Retry Trigger 1–34
About the Alarms extension 1–35
Alarms extension properties 1–35
About the Schedules extension 1–36
About the Point Manager 1–37
Points New Folder and New 1–38
Point Edit 1–39
About Point Discover, Add and Match (Learn Process) 1–41
About other Points views 1–45
About Histories extension views 1–46
History Import Manager 1–46
About the Device Histories View 1–49
About Schedules extension views 1–49
Schedule Import Manager 1–49
Schedule Import New 1–50
Schedule Import Edit 1–50
Schedule Import properties 1–51
Schedule Import or Export “gang” edits 1–52
About Schedule Import Discover, Add and Match (Learn Process) 1–52
Schedule Export Manager 1–53

Niagara Network 2–1

About the Niagara Network 2–1
NiagaraNetwork component notes 2–1
Niagara Tuning Policy notes 2–2
About the Fox Service 2–2
About History Policies 2–7
Niagara Station Manager notes 2–9
Station Learn and Discover notes 2–10
Station Add notes 2–12
NiagaraStation component notes 2–14
About station status properties 2–14
About client connection properties 2–14
About server connection properties 2–14
About station “provisioning” extensions 2–15
About the Users extension 2–15
About Users extension properties 2–15
Example Sync In and Out values 2–16
About Users sync strategy 2–17
Users device extension configuration notes 2–18
About the User Sync Manager 2–18
Sync process description 2–19
Niagara Discover enhancements 2–19
Niagara Point Manager notes 2–20
Niagara point Discover notes 2–20
About the Bql Query Builder 2–21
Niagara proxy point notes 2–24
Best practices for Niagara proxy points 2–25
Proxy of a proxy, other candidates 2–26
Link control and Niagara proxy points 2–26
NiagaraStation Alarms notes 2–27
Prepend and Append alarm routing notes 2–27

Station Schedules import/export notes 2–28
Niagara schedule import/export default configuration 2–28
Schedule Export Edit 2–28

Niagara histories notes 2–29

NiagaraStation Histories features 2–30

Niagara History Import Manager 2–30
Discovered selection notes 2–31
Niagara History Import properties 2–31
On demand properties in history import descriptors 2–32
Using System Tags to import Niagara histories 2–32

Niagara History Export Manager 2–33
Niagara History Export New 2–34
Niagara History Export Edit 2–34
Niagara History Export properties 2–35
About History Export Discover and Match (Learn Process) 2–36
Discovered selection notes 2–37
Using System Tags to export Niagara histories 2–37

About Niagara virtual components 2–38
Niagara virtuals background 2–38
Niagara virtuals in AX-3.7 2–38
Licensing and application overview 2–39
About the Niagara Virtual Device Ext 2–40
Ords for Niagara virtual components 2–42
Niagara virtuals cache (Virtual Policies) 2–42
Security and Niagara virtuals 2–45
Views on Niagara virtuals 2–46
Actions on Niagara virtuals 2–47
Niagara virtuals in Px views 2–47
Spy page diagnostics for Niagara virtual components 2–49
Localization support for Niagara virtuals 2–51

About Sys Def components 2–51
About the Niagara Sys Def Device Ext 2–51
About network level Sys Def components 2–52

About the Files extension 2–54
About the Niagara File Manager 2–54
Add or Edit dialog for NiagaraFileImports 2–55
About Niagara FileImport properties 2–56

Field Bus Integrations 3–1

Port and protocol variations 3–1
Ethernet-connected driver 3–1
Serial-connected driver 3–1
Special-port driver 3–2

Learn versus New devices and points 3–2

Serial tunneling 3–2
Serial tunnel overview 3–3
Client side (PC application) 3–3
To install the serial tunnel client on a Windows machine 3–3
Station side (TunnelService) 3–7
To configure the station for serial tunneling 3–7
To configure for safer tunneling access 3–8
Serial tunneling usage notes 3–9

Plugin Guides 4–1

Types of modules with plugins 4–1

Plugins in driver module 4–1
driver-DelimitedFileImportManager 4–1
driver-DeviceManager 4–2
driver-DriverManager 4–2
driver-FileDeviceManager 4–2
driver-HistoryExportManager 4–2
driver-HistoryImportManager 4–2
driver-PointManager 4–2

Plugins in niagaraDriver module 4–2
niagaraDriver-NiagaraFileManager 4–2
niagaraDriver-NiagaraHistoryExportManager 4–2
niagaraDriver-NiagaraHistoryImportManager 4–2
niagaraDriver-Niagara PointManager 4–2
niagaraDriver-NiagaraScheduleExportManager 4–3
niagaraDriver-NiagaraScheduleImportManager 4–3
niagaraDriver-ServerConnectionsSummary 4–3
niagaraDriver-StationManager 4–3
niagaraDriver-UserSyncManager 4–3

Plugins in niagaraVirtual module 4–3
niagaraVirtual-NiagaraVirtualCacheView 4–3
niagaraVirtual-Nva File View 4–3

Component Guides. 5–1

Component Reference Summary 5–1

Components in driver module 5–1
driver-ArchiveFolder 5–1
driver-ConfigRule 5–1
driver-DelimitedFileImport 5–2
driver-DriverContainer 5–2
driver-ExcelCsvFileImport 5–2
driver-FileDevice 5–3
driver-FileHistoryDeviceExt 5–3
driver-FileHistoryWorker 5–4
driver-FileNetwork 5–4
driver-HistoryNetworkExt 5–5
driver-HistoryPollScheduler 5–5
driver-PingMonitor 5–5
driver-SendTimer 5–5
driver-SendTimes 5–5
driver-TuningPolicy 5–5
driver-TuningPolicyMap 5–5

Components in fox module 5–5
fox-FoxClientConnection 5–5
fox-FoxFileSpace 5–6
fox-FoxServerConnection 5–6
fox-FoxService 5–6
fox-FoxSession 5–6
fox-ServerConnections 5–6

Components in niagaraDriver module 5–6
niagaraDriver-BogProvider 5–7
niagaraDriver-CyclicThreadPoolWorker 5–7
niagaraDriver-LocalSysDefStation 5–7
niagaraDriver-NiagaraAlarmDeviceExt 5–7
niagaraDriver-NiagaraFileDeviceExt 5–7
niagaraDriver-NiagaraFileImport 5–7
niagaraDriver-NiagaraFoxService 5–7
niagaraDriver-NiagaraHistoryDeviceExt 5–7
niagaraDriver-NiagaraHistoryExport 5–7
niagaraDriver-NiagaraHistoryImport 5–7
niagaraDriver-NiagaraNetwork 5–7
niagaraDriver-NiagaraPointDeviceExt 5–8
niagaraDriver-NiagaraPointFolder 5–8
niagaraDriver-NiagaraProxyExt 5–8

niagaraDriver-NiagaraScheduleDeviceExt 5–8
niagaraDriver-NiagaraScheduleImportExt 5–8
niagaraDriver-NiagaraStation 5–8
niagaraDriver-NiagaraStationFolder 5–8
niagaraDriver-NiagaraSysDefDeviceExt 5–8
niagaraDriver-NiagaraSystemHistoryExport 5–8
niagaraDriver-NiagaraSystemHistoryImport 5–9
niagaraDriver-NiagaraTuningPolicy 5–9
niagaraDriver-NiagaraTuningPolicyMap 5–9
niagaraDriver-NiagaraUserDeviceExt 5–9
niagaraDriver-NiagaraVirtualDeviceExt 5–9
niagaraDriver-ProviderStation 5–9
niagaraDriver-RoleManager 5–9
niagaraDriver-SyncTask 5–9

Components in niagaraVirtual module 5–9

niagaraVirtual-NiagaraVirtualBooleanPoint 5–10
niagaraVirtual-NiagaraVirtualBooleanWritable 5–10
niagaraVirtual-NiagaraVirtualCache 5–10
niagaraVirtual-NiagaraVirtualCachePolicy 5–10
niagaraVirtual-NiagaraVirtualComponent 5–10
niagaraVirtual-NiagaraVirtualDeviceExt 5–10
niagaraVirtual-NiagaraVirtualEnumPoint 5–10
niagaraVirtual-NiagaraVirtualEnumWritable 5–10
niagaraVirtual-NiagaraVirtualNumericPoint 5–10
niagaraVirtual-NiagaraVirtualNumericWritable 5–11
niagaraVirtual-NiagaraVirtualStringPoint 5–11
niagaraVirtual-NiagaraVirtualStringWritable 5–11

Components in serial module 5–11

serial-SerialHelper 5–11

Components in tunnel module 5–11

tunnel-TunnelService 5–11
tunnel-SerialTunnel 5–12
tunnel-TunnelConnection 5–12

Preface

Document Change Log

Document Change Log

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

- Updated: November 5, 2013
  Updated this document concurrent with the initial release of NiagaraAX-3.8, denoted in this document as "AX-3.8". This release provides additional security enhancements to those implemented in the 2013 "update 1" releases (AX-3.7u1). Document changes were mostly minor, as follows:
  - In the "About the Fox Service" section (in the "Niagara Network" main section) the section "Fox Service properties" on page 2-3 was edited, including a new subsection "FoxService defaults (new station) changed in AX-3.8" on page 2-6.
  - In the "About client connection properties" on page 2-14, a Note: about the non-portablility of client passwords was edited to mention the difference with an AX-3.8 station.
- Updated: June 5, 2013
  Updated this document concurrent with "update 1" release of AX-3.7, denoted in this document as "AX-3.7u1". This releases provide security enhancements that mainly affect the NiagaraNetwork (`niagaraDriver`). The following summarize the main changes made to this document:

  In the "Niagara Network" main section, the following subsections had changes or improvements:
  - "About the Fox Service" on page 2-2 was expanded, including "Fox Service properties" with a description of a new "Legacy Authentication" property starting in AX-3.7u1.
  - The "NiagaraStation component notes" section "About client connection properties" has a new Note: about the password property as "non-portable" in AX-3.7u1. Relating to the Users device extension, a new sub section "Update mismatch with Users device extension fault" on page 2-18 describes a condition that can occur while a system is being upgraded to AX-3.7u1.
  - The section "Best practices for Niagara proxy points" has a new subsection "Avoid links to proxy point actions" on page 2-25 that explains why this can result in issues.

  In the "Field Bus Integrations" main section, the "Serial tunneling" subsection had numerous security-related notes and cautions added, for example in the section "Serial tunnel overview" on page 3-3, "Serial tunnel client configuration" on page 3-5, and "Configuring the serial tunnel server" on page 3-7. A new section "Best security practices for tunneling" on page 3-8 provides related details.
- Updated July 22, 2012
  Document is a "version-split" from previous versions, and applies to NiagaraAX-3.7 (AX-3.7), especially in regards to the "niagaraDriver", as related changes were made in AX-3.7. With this version split, some notes and references to the previous (AX-3.6 and AX-3.5) revisions remain, but most references to the oldest revisions (AX-3.0 - AX-3.4) are gone.

  AX-3.7-related changes to this document are many, but are concentrated in the following sections:
  - "Niagara Network" changes includes sections "About the Fox Service" on page 2-2 including subsection "Fox Service properties" on page 2-3.
    The main section on "About Niagara virtual components" on page 2-38 was reworked to reflect changes in the Niagara virtuals architecture, with most screen captures updated. New subsections include "Niagara virtuals background" on page 2-38, "Niagara virtuals in AX-3.7" on page 2-38, "About the Niagara Virtual Device Ext" on page 2-40, and "Niagara virtuals cache (Virtual Policies)" on page 2-42.
  - In the "Field Bus Integrations" section, the subsection "Serial tunneling" on page 3-2 was updated to describe new self-executable files for installing the NiagaraAX serial tunneling client on

Windows 7 and Windows Vista PCs. Included are details on installed files and the Windows interface for all variations of the serial tunneling client.

- Some new entries were added to the summary descriptions in the "Component Guides" and "Plugin Guides" to support context sensitive Help in Workbench, mostly relating to Niagara virtuals.

CHAPTER 1

Driver architecture

In any NiagaraAX station, one or more driver networks are used to fetch and model real-time data values. Real-time values are modeled with proxy points, *lower-tier* components in that driver's architecture. In addition, (depending on the driver/devices) other data items "native" in devices may also integrated into the station, such as schedules, alarms, and data logs (histories).

To support proxy points and other modeled data, the station must have that driver's *network architecture*.

The following main topics apply to common driver architecture:

- What are networks?
- About Network architecture
- About the Driver Manager
- Common network components
- About the Device Manager
- Common device components
- Virtual gateway and components
- Types of device extensions
- About the Point Manager
- About Histories extension views
- About Schedules extension views

What are networks?

Networks are the top-level component for any NiagaraAX driver. For drivers that use field bus communications, such as Lonworks, BACnet, and Modbus (among many others), this often corresponds directly to a *physical network* of devices. Often (except for BACnet), the network component matches one-to-one with a specific comm port on the NiagaraAX host platform, such as a serial (RS-232 or RS-485) port, Lonworks FTT-10 port, or Ethernet port.

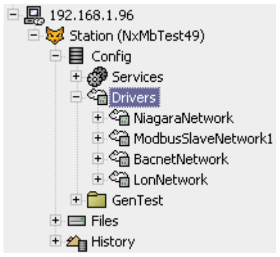
Note: *A BacnetNetwork component is unique because it supports multiple logical BACnet networks, which sometimes use different comm ports (e.g. if a JACE with BACnet MS/TP, one or more RS-485 ports). See the* Bacnet Guide *for more details.*

Other "non field bus" drivers *also* use a network architecture, for example the Ndio driver (Niagara Direct Input / Output) and Nrio driver (Niagara Remote Input/Output) each have a network (NdioNetwork, NrioNetwork, respectively) to interface to physical I/O points on the appropriate host JACE or hardware I/O module (either directly attached or remotely connected). Also, "database" drivers also use a network architecture, for example the "rdbSqlServer" driver includes an RdbmsNetwork. (Database drivers apply only to Supervisor or AX SoftJACE hosts.)

Where are networks located?

In any NiagaraAX station, *all* driver networks are located under the DriverContainer component in the station's database ("Drivers" in the root of the station, as shown in Figure 1-1).

Figure 1-1 *DriverContainer in Nav tree*



Note: *By default, most stations include a NiagaraNetwork. It may be used to model data in other NiagaraAX stations. Also, it contains the NiagaraFoxService required for Workbench-to-station communications. For more details, see "Niagara Network" on page 2-1.*

Note: *By convention, you should keep all driver networks located under the station's Drivers container—it is a special component with a view that provides some utility. See "About the Driver Manager" on page 1-4 for more details.*

About Network architecture

To represent any driver in a station database, a consistent "driver framework" using a network architecture is used. This includes the upper-tier parent *network* component, and one or more child *device* components, each with *device ext* (extension) child components.

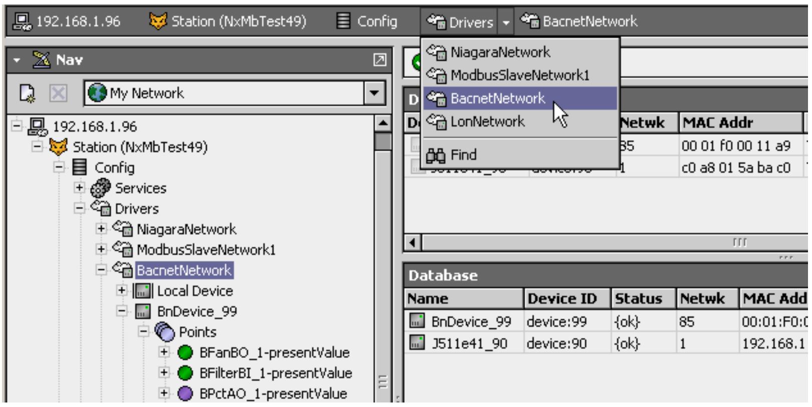
The following sections explain further:

- "Network component hierarchy"
- "About network components"
- "About device components"
- "About device extensions"

Network component hierarchy

Hierarchically, the component parentage is: network, device, device extensions, points (Figure 1-2).

Figure 1-2 *Example driver architecture (Bacnet)*



Name	Device ID	Status	Netwk	MAC Add
BnDevice_99	device:99	{ok}	85	00:01:F0:0
J511e41_90	device:90	{ok}	1	192.168.1

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

- If engineering a JACE station, you invariably add *additional* driver networks, opening the driver's palette and copying the network-level component into Drivers. Or, you can use the New button in the Driver Manager. Examples are a LonNetwork or BacnetNetwork.
- If engineering a Supervisor (PC) station, the NiagaraNetwork may be the only one needed. Optionally, you may add a "database" network (providing the PC host is licensed for it). And, if a PC host licensed as a direct "integration Supervisor" (e.g. BACnet Supervisor), you may add a driver network (e.g. BacnetNetwork) directly into its Drivers container.

Note: *Regardless of host platform, in most cases the network component is the only item you need to manually copy from that driver's palette. After that, you use "manager views" to add child components like devices and proxy points, even if programming offline.*

This enforces the correct component hierarchy under that driver as you engineer, and helps coordinate mechanisms used by the driver for communications. Exceptions to this rule are noted (as applicable) in other documentation about NiagaraAX drivers.

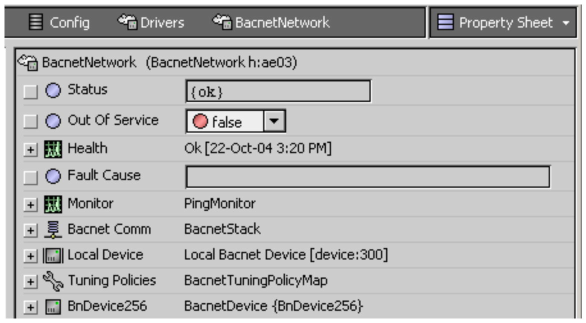
About network components

A network component ("generically": DriverNetwork) is the top-level component for any driver, and by default has mandatory (frozen) component slots, such as:

- "Health" and other status properties—see "Network status properties" on page 1-6.
- AlarmSourceInfo—see "About network Alarm Source Info" on page 1-6.
- PingMonitor—see "About Monitor" on page 1-7.
- TuningPolicyMap—see "About Tuning Policies" on page 1-8.

Access these slots in the DriverNetwork's property sheet (Figure 1-3).

Figure 1-3 *Example BacnetNetwork property sheet*



BacnetNetwork (BacnetNetwork h:ae03)

Status	{ok}
Out Of Service	false
Health	Ok [22-Oct-04 3:20 PM]
Fault Cause	
Monitor	PingMonitor
Bacnet Comm	BacnetStack
Local Device	Local Bacnet Device [device:300]
Tuning Policies	BacnetTuningPolicyMap
BnDevice256	BacnetDevice {BnDevice256}

Note: *Depending on the specific driver, there may be additional slots in the DriverNetwork. For example, if a field bus network, there may be "PollScheduler" and/or "CommConfig" slots. Some of these may require proper configuration before driver communications occur. See "Additional network components" on page 1-13.*

The DriverNetwork is also the *parent container* for all device-level components. Devices list in tabular format in the default view for that network, the DeviceManager. See "About the Device Manager" on page 1-14. You use the Device Manager to create and manage device components, and (if the driver provides this), use discover features.

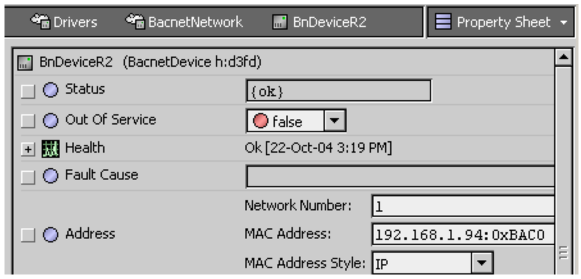
About device components

A device component ("generically": DriverDevice) is a second-tier component for any driver, and by default has mandatory (frozen) component slots, such as:

- "Health" and other status properties—see "Device status properties" on page 1-22.
  Note: *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. See "Driver-specific device slots" on page 1-23. Or, a device may have a "Virtual" gateway slot. See "Virtual gateway and components" on page 1-23.*
- One or more *device extensions*—for example "Points" (DriverDevicePointExt). See "About device extensions" on page 1-4.

Typically, the property sheet is the default view for a device component—you can access device properties, slots, and its extensions using the device's property sheet (Figure 1-4).

Figure 1-4 *Example BacnetDevice property sheet*

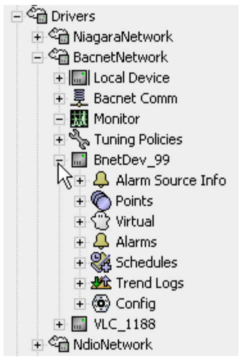


BnDeviceR2 (BacnetDevice h:d3fd)

Status	{ok}
Out Of Service	false
Health	Ok [22-Oct-04 3:19 PM]
Fault Cause	
Address	Network Number: 1 MAC Address: 192.168.1.94:0xBAC0 MAC Address Style: IP

The DriverDevice is also the *parent container* for all device extensions. As shown in Figure 1-5, device extensions (e.g. "Points") are visible under the device when you expand it in the nav tree.

Figure 1-5 *Example BacnetDevice extensions in Nav tree*



About device extensions

A device extension is a child of a device, and represents some functionality of the device. Each extension contains properties and other components. Device extensions are *container* components, often with one or more *special views*. For more details, see "Types of device extensions" on page 1-26.

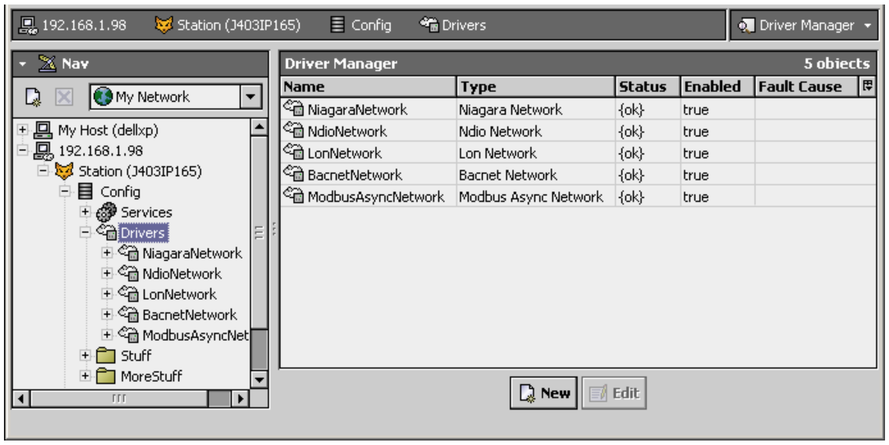
For example, in any of the field bus networks, each device has a **`Points`** extension, the parent container for proxy points. The default view of the Points extension is the **`Point Manager`**, which you use to create and manage proxy points. See "About the Point Manager" on page 1-37 for details.

Note: *Device extensions are required (frozen) components of a device—you cannot delete them. They are created automatically when you add a device to the station database. This varies from the "point extension" model, where you individually add/delete extensions under control points and components.*

About the Driver Manager

The Driver Manager is the default view for the DriverContainer (showing all networks) in a station. Figure 1-6 shows an example Driver Manager for a JACE station.

Figure 1-6 *Driver Manager*



Driver Manager — 5 objects

Name	Type	Status	Enabled	Fault Cause
NiagaraNetwork	Niagara Network	{ok}	true	
NdioNetwork	Ndio Network	{ok}	true	
LonNetwork	Lon Network	{ok}	true	
BacnetNetwork	Bacnet Network	{ok}	true	
ModbusAsyncNetwork	Modbus Async Network	{ok}	true	

By default, each driver network lists showing its name, a type description, a real-time status field, whether it is enabled, and a fault cause field (empty, unless the network is in fault).

Within the Driver Manager view, network-level operations are available:

- Double-click a network to go to its Device Manager view. See "About the Device Manager" on page 1-14.
- Right-click a network to see its popup menu, including available actions. Available actions will vary by driver, but may include "Ping," "Upload," and "Download" as examples.

Buttons at the bottom of the Driver Manager provide other functions. See the section "Driver Manager New and Edit" on page 1-4.

Driver Manager New and Edit

Buttons at the bottom of the Driver Manager include:

- New

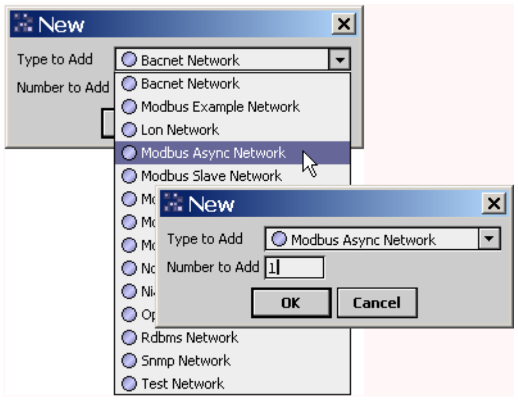
- Edit

Note: *New and Edit are also on the Driver Manager toolbar and the Manager menu.*

New

The New button in the Driver Manager allows you to add a new driver network to the station. This is equivalent to copying a driver network component from that driver's palette. The New dialog provides a selection list for network type, and also the number of networks to add (Figure 1-7).

Figure 1-7 *New dialog in Driver Manager*

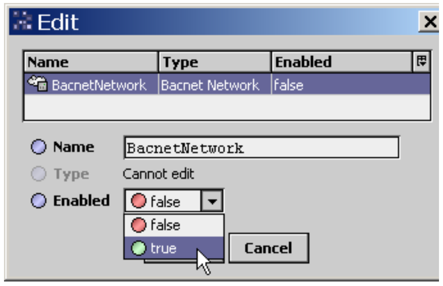


Note: *Currently, you are allowed to add multiples of any driver network, as well as any network type (regardless of the station's host platform). However, please understand that many networks should be "one-only" per station, e.g. a NiagaraNetwork and BacnetNetwork. Also, consider that networks shown in the selection list reflect only the modules available on your Workstation, and may not be present (or supported) on the target NiagaraAX host.*

Edit

The Edit button in the Driver Manager allows you to rename a selected network, or to set it to disabled (Figure 1-8).

Figure 1-8 *Edit dialog in Driver Manager*



Caution *Whenever you set a network to disabled (Enabled = false), the network component and all child driver components (all devices, proxy points, etc.) change to disabled status. If a field bus driver, communications routines like polling and device status pings also suspend. By default, disabled status color is gray text on a light gray background.*

This network-wide action may be useful for maintenance purposes. Note that Enabled is also individually available at the device-level and proxy point-level, using the Edit button/dialog in the Device Manager or Point Manager, respectively.

A disable at the device-level disables all child (proxy points), as well as polling to that points under that device. A disable at the proxy-point level disables the single point.

Common network components

Each driver network contains common properties and other components. This section describes those items.

- Network status properties
- About network Alarm Source Info
- About Monitor
- About Tuning Policies
- Additional network components

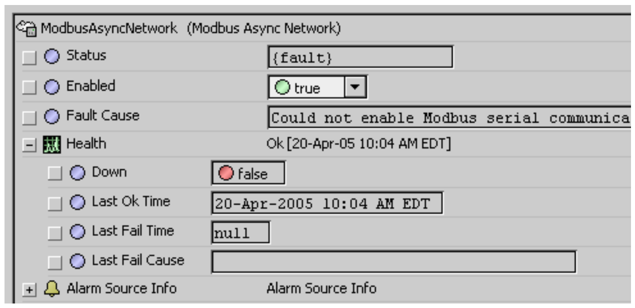
Network status properties

Status

The Status property of a network indicates if the driver is capable of communications—for any configured driver, network Status is typically "{ok}." However, when adding some networks, the initial Status may be fault, as shown in Figure 1-9. This might mean a communications port is yet unassigned, or there is a port conflict.

Note: *A fault status also occurs if the host platform is not properly licensed for the driver being used. If a network fault, see the* Fault Cause *property value for more details.*

Figure 1-9 *Network status properties*



ModbusAsyncNetwork (Modbus Async Network)

Status	{fault}
Enabled	true
Fault Cause	Could not enable Modbus serial communica
Health	Ok [20-Apr-05 10:04 AM EDT]
Down	false
Last Ok Time	20-Apr-2005 10:04 AM EDT
Last Fail Time	null
Last Fail Cause	
Alarm Source Info	Alarm Source Info

Enabled

By default, network Enabled is true—you can toggle this in the property sheet, or in the Driver Manager (by selecting the network and using the Edit button). See related Caution on page 5.

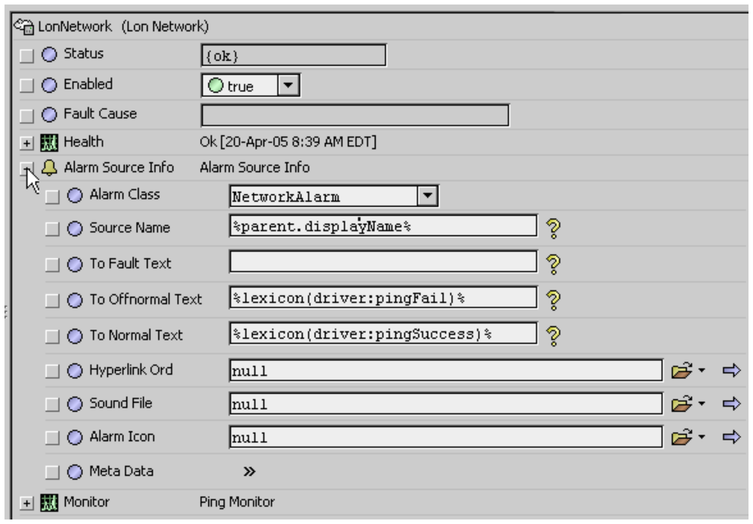
Health

Network Health contains historical properties about the relative health of the network in the station, including historical timestamps.

About network Alarm Source Info

A network's Alarm Source Info slot holds a number of common alarm properties (Figure 1-10). These properties are used to populate an alarm if the *network* does not respond to a monitor ping. See "About Monitor" for details on the monitor mechanism.

Figure 1-10 *Example Network Alarm Source Info properties*



LonNetwork (Lon Network)

Status	{ok}
Enabled	true
Fault Cause	
Health	Ok [20-Apr-05 8:39 AM EDT]
Alarm Source Info	Alarm Source Info
Alarm Class	NetworkAlarm
Source Name	%parent.displayName%
To Fault Text	
To Offnormal Text	%lexicon(driver:pingFail)%
To Normal Text	%lexicon(driver:pingSuccess)%
Hyperlink Ord	null
Sound File	null
Alarm Icon	null
Meta Data	»
Monitor	Ping Monitor

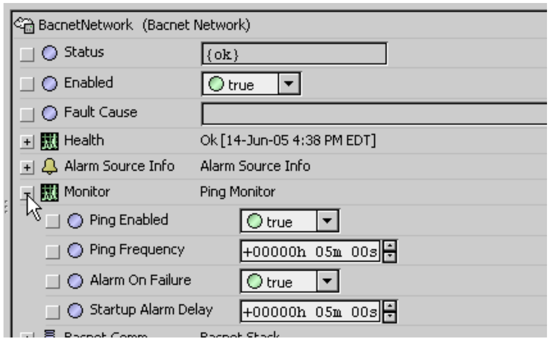
Alarm Source Info properties work the same as those in an alarm extension for a control point. For property descriptions, see the *User Guide* section "About alarm extension properties".

Note: *Each child device object also has its own Alarm Source Info slot, with identical (but independently maintained) properties. See "Device Alarm Source Info" on page 1-22.*

About Monitor

A network's Monitor slot holds configuration for the "ping mechanism" used by the driver network. In the network's property sheet, expand the Monitor slot to see configuration (Figure 1-11).

Figure 1-11 *Example Monitor properties*



BacnetNetwork (Bacnet Network)
Status {ok}
Enabled true
Fault Cause
Health Ok [14-Jun-05 4:38 PM EDT]
Alarm Source Info Alarm Source Info
Monitor Ping Monitor
Ping Enabled true
Ping Frequency +00000h 05m 00s
Alarm On Failure true
Startup Alarm Delay +00000h 05m 00s

Monitor provides verification of the general health of the network, plus the network's "pingables" (typically, devices) by ensuring that each device is minimally "pinged" at some repeating interval.

- If a device responds to the monitor ping, device status is typically "ok," and normal communications routines to it (proxy-point polling, plus reads of device schedules, trends, etc. if supported by the driver) proceeds normally. Typically, this applies even if the device returns an error response to the ping, because this indicates that the device is "alive."
- If a device does not respond to the monitor ping, it is marked with a *down* status—this causes normal communications routines to that device to be suspended. Upon the next successful monitor ping to that device, device status typically returns to "ok" and normal communications routines resume.

Note: *Whenever successful communications occur to a device, that device component's* Health *property is updated with the current time. The network ping Monitor will only "ping" 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, providing 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, a "ping fail" condition is immediately noted, without waiting for the monitor ping interval.*

The following sections provide more Monitor details:

- Monitor properties
- Monitor considerations by driver

Monitor properties

The monitor ping properties are as follows:

- **Ping Enabled**
  - If true, (default) a ping occurs for each device under the network, as needed.
  - While set to false, device status pings do not occur. Moreover, device statuses cannot change from what existed when this property was last true.

  Note: *It is recommended you leave Ping Enabled as true in almost all cases.*
- **Ping Frequency**
  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**
  - If true (default), an alarm is recorded in the station's AlarmHistory upon each ping-detected device event ("down" or subsequent "up").
  - If false, device "down" and "up" events are not recorded in the station's AlarmHistory.
- **Startup Alarm Delay**
  Specifies the period a station must wait after restarting before device "down" or "up" alarms are generated. Applies only if the Monitor's property Alarm On Failure is true.

Monitor considerations by driver

The monitor mechanism used by a specific driver may have unique characteristics.

- For example, in a BacnetNetwork, any monitor ping is directed to the device's BACnet "Device Object," and in particular, to its "System_Status" property. In this unique case, a received response of "non-operational" is evaluated the same as getting no response at all!
- Or, in any Modbus network, when a monitor ping message is sent, it is directed to the device's "Ping Address," which is configured by several properties in the ModbusDevice object.

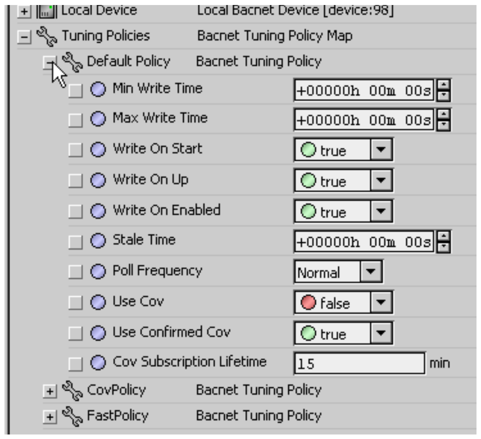
Other drivers may have specific considerations for the Monitor ping mechanism. For more information, refer to the "Device Monitor Notes" section within any NiagaraAX driver document.

About Tuning Policies

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 (Figure 1-12).

Figure 1-12 *Example Tuning Policies Map (Bacnet)*



Local Device	Local Bacnet Device [device:98]
Tuning Policies	Bacnet Tuning Policy Map
Default Policy	Bacnet Tuning Policy
Min Write Time	+00000h 00m 00s
Max Write Time	+00000h 00m 00s
Write On Start	true
Write On Up	true
Write On Enabled	true
Stale Time	+00000h 00m 00s
Poll Frequency	Normal
Use Cov	false
Use Confirmed Cov	true
Cov Subscription Lifetime	15 min
CovPolicy	Bacnet Tuning Policy
FastPolicy	Bacnet Tuning Policy

Note: *Some driver networks do not have Tuning Policies, for example an RdbmsNetwork for a database driver. Also, the NiagaraNetwork has greatly simplified Tuning Policies.*

By default, a driver's TuningPoliciesMap contains just a single TuningPolicy ("Default Policy"). However, you typically 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 Niagara 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 4 available (and different) tuning policies to pick from when you create and edit proxy points, where you could selectively apply the policy needed.

The following sections provide more Tuning Policy details:

- Tuning Policy properties

- Tuning Policy considerations by driver

Tuning Policy properties

Tuning Policy properties for typical field bus drivers are as follows:

- **Min Write Time**
  Applies to writable proxy points, especially ones that have one or more linked inputs. Specifies the minimum amount of time allowed between writes. Provides a method to throttle rapidly changing value so that only the last value is written. If this property value is 0 (default), this rule is disabled (all value changes attempt to write).
- **Max Write Time**
  Applies to writable proxy points. Specifies the maximum "wait time" before *rewriting* the value, in case nothing else has triggered a write. Any write action resets this timer. If property value is 0 (default), this rule is disabled (no timed rewrites).
  Note: *In some cases setting this to some value, for example 10 minutes, may be useful. Often, a network may have devices that upon a power cycle (or even a power "bump"), have writable points that reset to some preset "default" value or state. Note that often in a "site-wide" power bump of a few seconds, such field controllers (devices on the network) typically reset, but a JACE continues normal operation on 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.*

  Here, 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 issues like this.

  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. Exercise caution with this property in this case, to avoid "write contention" that could result in toggling loads.
- **Write On Start**
  Applies to writable proxy points. Determines behavior at *station startup*.
  - If true, (default) a write occurs when the station first reaches "steady state."
  - If set to false, a write does not occur when the station reaches "steady state."

  Note: *Consider setting this to false in 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), will upon station startup attempt to write to all 4,000 points, putting a significant load on the station. As a consequence, it is possible that in this scenario the Bacnet driver (network) may generate "write queue overflow" exceptions.*
- **Write On Up**
  Applies to writable proxy points. Determines behavior when proxy point (and parent device) transitions from "down" to "up."
  - If true, (default) a write occurs when the parent device transitions from down to up.
  - If set to false, a write does not occur when the parent device transitions from down to up.
- **Write On Enabled**
  Applies to writable proxy points. Determines behavior when a proxy point's status transitions from "disabled" to normal (enabled).
  - If true, (default) a write occurs when writable point transitions from disabled.
  - If set to false, a write does not occur when writable point transitions from disabled.

  Note: *The disabled-to-enabled status transition can be inherited globally by points if the parent device had been set to disabled—or network-wide if the driver network was set to disabled. Therefore, be aware that if left at* `true` *in tuning policies, that all associated writable points receive a write upon either the device or network when it transitions from status "disabled" to "enabled."*
- **Stale Time**
  Applies to all proxy points.
- If set to a non-zero value, points become "stale" (status stale) if the configured time elapses without a successful read, indicated by Read Status "ok."
- If set to zero (default), the stale timer is disabled, and points become stale immediately when unsubscribed.
  By default, proxy point status "stale" is indicated by tan background color. In addition, stale status is considered "invalid" for any downstream-linked control logic. For more details, see the *User Guide* section "About "isValid" status check".

Note: *Stale time is recommended to be specified 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 for some reason, then another poll cycle time or two is allowed for the message to be received before setting the stale flag.*

- **Poll Frequency**
  (May not exist in some driver's tuning policies, but is instead a separate property of each ProxyExt) Applies to all proxy points. Provides a method to associate the tuning policy with one of 3 Poll Rates available in the network's Poll Service: Fast Rate, Normal Rate, or Slow Rate. The default poll frequency is "Normal."
  Note: *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.*

Tuning Policy considerations by driver

Tuning policies used by a specific driver may have unique characteristics. For example, under a NiagaraNetwork, its TuningPolicy has only three properties: Stale Time, Min Update Time, and Max Update Time. For more details, see "NiagaraStation component notes" on page 2-14.

Other drivers may have specific considerations for tuning policies. For more information, refer to the "Tuning Policy Notes" section within any NiagaraAX driver document.

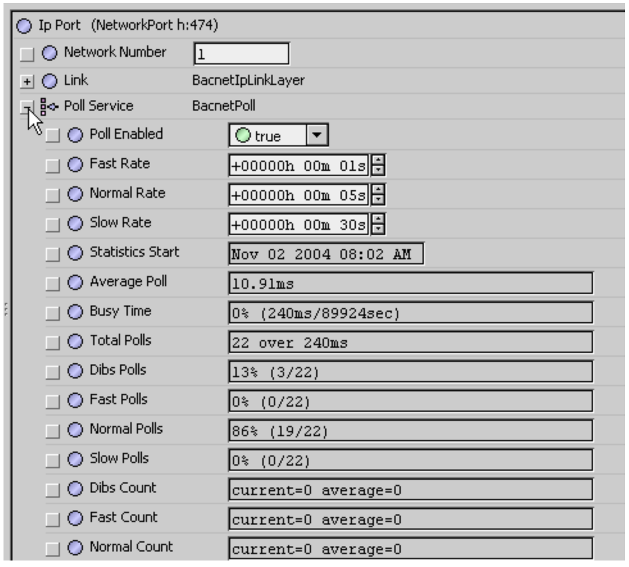
About poll components

Many driver networks use a single polling mechanism (Poll component, named "Poll Service" or "Poll Scheduler") adjusted at the network level in the station. In the case of a BacnetNetwork, a separate BacnetPoll is used for each port under the BacnetStack's Network component, as shown in Figure 1-13.

- Poll scheduler operation
- Poll Service properties
- Using poll statistics in tuning poll rates

Note: *The Niagara Network does not use polling, but uses subscriptions only.*

Figure 1-13 *Example BacnetPollService*



Ip Port (NetworkPort h:474)

Network Number	1
Link	BacnetIpLinkLayer
Poll Service	BacnetPoll
Poll Enabled	true
Fast Rate	+00000h 00m 01s
Normal Rate	+00000h 00m 05s
Slow Rate	+00000h 00m 30s
Statistics Start	Nov 02 2004 08:02 AM
Average Poll	10.91ms
Busy Time	0% (240ms/89924sec)
Total Polls	22 over 240ms
Dibs Polls	13% (3/22)
Fast Polls	0% (0/22)
Normal Polls	86% (19/22)
Slow Polls	0% (0/22)
Dibs Count	current=0 average=0
Fast Count	current=0 average=0
Normal Count	current=0 average=0

Poll scheduler operation

Regardless of driver type, the Poll Service provides a "poll scheduler" that operates in the same basic manner.

Note: *A driver typically uses a single thread for all polling on that specific network. However, there are some exceptions to this model. For example, polling in a BacnetNetwork uses two threads per network port. It is important to note that if a driver's Poll Service uses multiple threads, that poll statistics reflect the sum of activity for all the threads—there is no way to determine a statistics breakdown for each thread.*

- The Poll Service provides three different configurable poll rates (Slow, Normal, Fast). Note these are just arbitrary names—there is no logic that enforces a relationship between their values. Meaning, the slow rate can actually be configured for a faster interval than the normal rate, without issues.

- "Pollables" (mainly proxy points) are associated with one of three "rate" groups (Slow, Normal, Fast) via either:
  - assigned Tuning Policy in each point's proxy extension (BacnetNetwork)
  - assigned Poll Frequency in each point's proxy extension (most drivers other than Bacnet)

  In the case of device objects, a Poll Frequency property is used to select the rate directly.
- The poll scheduler maintains a group of *four* rate "buckets" to service "pollables", three of which correspond to these configured poll rates (slow, normal, fast).
  A fourth "dibs stack" bucket is allocated for 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 permanently subscribed point is first polled (thereafter, it is no longer "dibs stack-polled").
- Every 10 seconds, the poll scheduler rebuilds the list of objects that are assigned to each of the poll buckets. An algorithm is used to break up the list of objects for each poll bucket into optimally-sized groups, which allows the poll thread to switch back and forth between the rate buckets. This algorithm is designed to spread the message traffic out evenly over the configured intervals, and to allow points assigned to quicker buckets to update multiple times before points assigned to a slower bucket update once.
- Poll statistics are updated every 10 seconds. Fast, slow, and normal cycle times display the average time in milliseconds (ms) to complete a single poll cycle. The poll scheduler algorithm automatically calculates an inter-message delay time to evenly spread poll messages out over the configured rate. For example, if there are 5 points assigned to a normal rate/Tuning Policy, then it may poll a point in the list every 2 seconds. In this case, the normal poll cycle time would be around 10000 ms, but that does not mean that actually took 10 seconds to actually poll those 5 points.

Priority of polling by the scheduler occurs in this fashion:

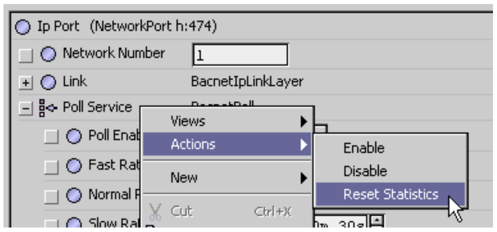
1. Dibs stack. When first subscribed, a pollable moves to the top of the dibs stack (first dibs). The poll scheduler always polls the dibs bucket before doing anything else. The dibs stack is polled last-in, first-out (LIFO). As long as entries are in the dibs stack, they are polled as fast as possible with no delays.
2. When the dibs stack is empty, the scheduler attempts to poll the components in each "rate" bucket using an algorithm designed to create uniform network traffic.

   For example, if the fast rate is configured to 10 seconds and there are 5 components currently subscribed in the fast bucket, then the scheduler will attempt to poll one component every 2 seconds.

Note: *Every ten seconds the poll scheduler rechecks the buckets for configuration changes. So if a pollable's configuration is changed from slow to fast, it takes at most ten seconds for the change to take effect.*

You can manually reset poll statistics using the Reset Statistics action to the Poll Service (Figure 1-14).

Figure 1-14 *Reset Statistics action of PollService*



Poll Service properties

Properties of the Poll Service component for typical field bus drivers include four writable properties and various read-only statistics properties, as follows:

- **Poll Enabled**
  - If true (default), polling occurs for all associated pollables (proxy points, devices) under the network component, or if a BacnetPoll, under that BacnetStack, Network, Port.
  - While set to false, polling is suspended and further value updates from polling do not occur.

  Note: *PollService actions Enable and Disable allow access to this property, see Figure 1-14.*

  Note: *The three Rate properties below are named arbitrarily, however, typical convention is to use them as named. For related information, see "Poll scheduler operation" on page 1-10.*
- **Fast Rate**
  Target poll interval for pollables assigned to this rate group (default often is 1 second).

- **Normal Rate**
  Target poll interval for pollables assigned to this rate group (default often is 5 seconds).
- **Slow Rate**
  Target poll interval for pollables assigned to this rate group (default often 30 seconds).
  Note: *All remaining properties are read-only statistics properties.*
- **Statistics Start**
  Timestamp reflecting 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**
  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 rate buckets. It is the time spent polling a given group before pausing and switching to another group of objects, either in the same or a different poll rate bucket.
- **Busy Time**
  Displays a *percentage* of time spent by the poll thread actually polling points, across all poll buckets. Includes (in parentheses) a ratio of "(*time spent polling/total time since statistics were restarted*)". Given a small amount of time is spent transitioning between poll buckets, 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. Also see "Using poll statistics in tuning poll rates" on page 1-13.
  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 the Busy Time in half to get an average busy time for each thread.*
- **Total Polls**
  Total number of polls performed, along with the time spent actually polling. Note that 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 any object has been polled. It is not a running total of the actual poll cycles.

Total Polls, Example You have 2 points assigned to a fast policy of 1 second, 2 points assigned to a normal policy of 5 seconds, and 2 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

Note: *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 if performing "read property multiple" for BACnet devices, or possibly when performing COV subscriptions.*

- **Dibs Polls**
  Percentage and ratio of the number of dibs polls versus the total polls.
- **Fast Polls**
  Percentage and ratio of the number of fast polls versus the total polls.
- **Normal Polls**
  Percentage and ratio of the number of normal polls versus the total polls.
- **Slow Polls**
  Percentage and ratio of the number of slow polls versus the total polls.
- **Dibs Count**
  Current and average number of components in dibs stack.
- **Fast Count**
  Current and average number of components in fast queue.
- **Normal Count**
  Current and average number of components in normal queue.
- **Slow Count**
  Current and average number of components in slow queue.
- **Fast Cycle Time**
  Average cycle time of the fast queue.
- **Normal Cycle Time**
  Average cycle time of the normal queue.
- **Slow Cycle Time**
  Average cycle time of the slow queue.

Note: *Depending on 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 document for unique Poll Service features or notes.*

Using poll statistics in tuning poll rates

There are several statistics (Poll Service status properties) to watch when setting poll rate times and/or assigning points to different poll frequencies/tuning policies. See the related sections "Poll scheduler operation" on page 1-10 and "Poll Service properties" on page 1-11 for detailed descriptions.

Note: *Again, it is important to note that if a driver's Poll Service uses multiple threads (such as in the Bacnet driver), that poll statistics reflect the sum of activity for all the threads—there is no way to determine a statistics breakdown for each thread.*

- Busy Time - How busy is too busy?
- Tuning poll rates

Busy Time - How busy is too busy?

A Poll Service with a busy time being 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 0 seconds.

Note: *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 which case the Poll Service will use that setting as its minimum. In these cases, the inter-message delay may be longer than the configured minimum, but it will never be less than the minimum time.*

Note when points are first subscribed, they are initially added to the dibs poll bucket for immediate polling. After this initial poll, they are moved to their assigned poll buckets, which simply results in a longer poll cycle time for that rate bucket.

Tuning poll rates

Typically, there is a "base number" of points that are permanently subscribed, including any points that are linked, and/or have a history or alarm extension. Note these permanently subscribed points will always reflect in their assigned poll rate queue. A good starting point would be to adjust the configured poll rates such that the Busy Time is at 75%, with only these points polling.

Maintaining some free time for the Poll Service should allow subscription surges (users viewing 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, then it may be necessary to adjust the configured poll rates, such that steady state Busy Time is below 75%.

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

If a bucket's actual poll cycle time is longer than the configured time, then this indicates that 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 effects on the Busy Time, but it will not speed up the polling process either—unless the quantity of points in that queue is reduced.

If the JACE's CPU usage is high, and the poll service's Busy Time is near 100%, then increasing the 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. The Poll Service's Busy Time and poll cycle times do not reflect any statistics regarding points that update values using these other methods, such as nvUpdates or any similar mechanisms.*

Additional network components

Depending on the driver type, a network typically includes other components that define communications settings. Access these components on the property sheet of the network. More details are in the following subsections:

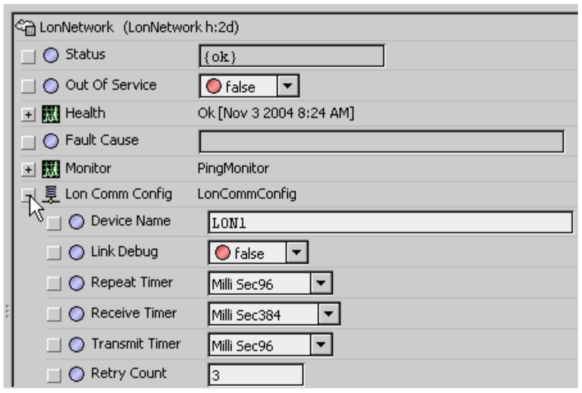
- About communication components
- About driver-specific properties

About communication components

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

For example, the NiagaraNetwork contains the NiagaraFoxService, a BacnetNetwork contains a Bacnet-Stack (Bacnet Comm), a LonNetwork contains LonCommConfig (Figure 1-15).

Figure 1-15 *LonNetwork's LonCommConfig component*



LonNetwork (LonNetwork h:2d)

Status	{ok}
Out Of Service	false
Health	Ok [Nov 3 2004 8:24 AM]
Fault Cause	
Monitor	PingMonitor
Lon Comm Config	LonCommConfig
Device Name	LON1
Link Debug	false
Repeat Timer	Milli Sec96
Receive Timer	Milli Sec384
Transmit Timer	Milli Sec96
Retry Count	3

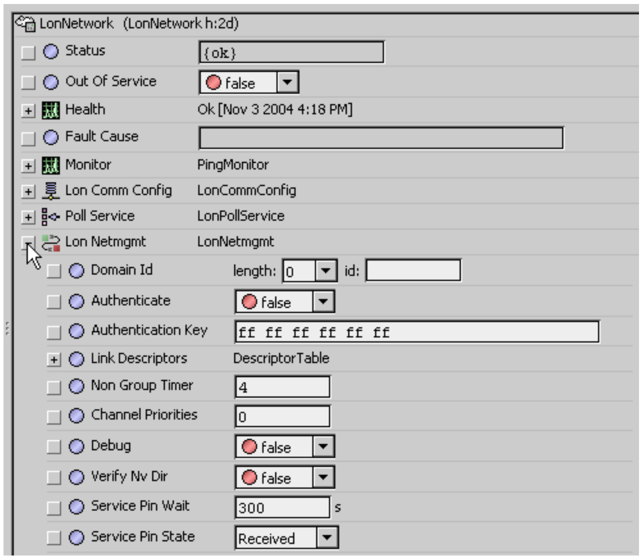
See the driver document for specific details on configuring a network's communications components.

About driver-specific properties

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

For example, the NiagaraNetwork has container components Fox Service and History Policies, a ModbusAsyncNetwork contains a number of properties for timeouts and byte-order conventions, and a LonNetwork contains a LonNetmgt component with various child properties (Figure 1-16).

Figure 1-16 *LonNetwork's LonNetmgmt component*



LonNetwork (LonNetwork h:2d)

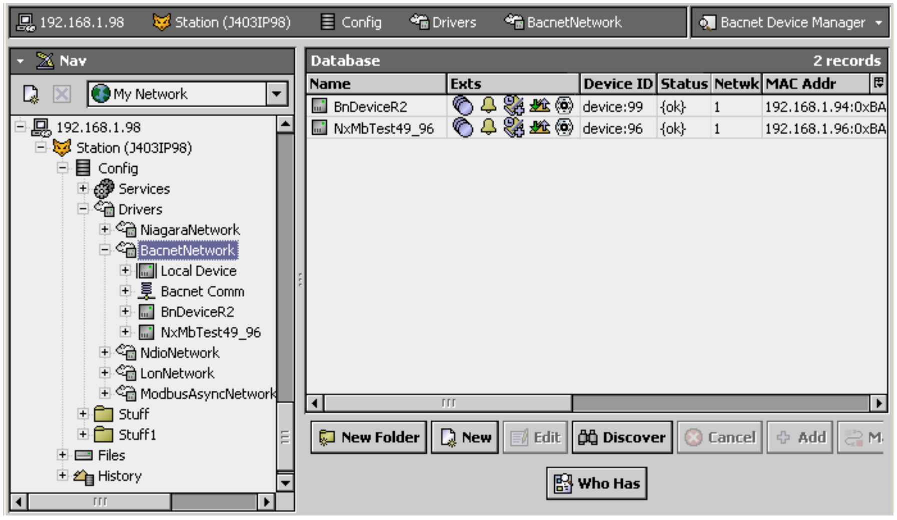
Status	{ok}
Out Of Service	false
Health	Ok [Nov 3 2004 4:18 PM]
Fault Cause	
Monitor	PingMonitor
Lon Comm Config	LonCommConfig
Poll Service	LonPollService
Lon Netmgmt	LonNetmgmt
Domain Id	length: 0 id:
Authenticate	false
Authentication Key	ff ff ff ff ff ff
Link Descriptors	DescriptorTable
Non Group Timer	4
Channel Priorities	0
Debug	false
Verify Nv Dir	false
Service Pin Wait	300 s
Service Pin State	Received

For more details see "Niagara Network" on page 2-1, or the corresponding driver document for driver-specific properties and components of a network.

About the Device Manager

The Device Manager 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 (Figure 1-17). When building a network in the station, you use this view to create, edit, and delete device-level components.

Figure 1-17 *Example Device Manager (BacnetDeviceManager)*



Name	Exts	Device ID	Status	Netwk	MAC Addr
BnDeviceR2		device:99	{ok}	1	192.168.1.94:0xBA
NxMbTest49_96		device:96	{ok}	1	192.168.1.96:0xBA

Following station configuration, this view provides a status and configuration summary for all devices networked 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 (see “Types of device extensions” on page 1-26 for details). You can also use it to issue actions to selected devices, such as “ping,” upload, and so forth.

At the bottom of the view, buttons “New Folder” and “New” let you create new device folders and devices in the station database. The “Edit” button lets you edit one or more selected devices in the station database. If the driver supports online “device learning,” buttons “Discover,” and “Match” are also typically available. Finally, additional buttons may be available, depending on the driver.

For more details, see:

- Device New Folder and New
- All Descendants
- Device Edit
- About Device Discover, Add and Match (Learn Process)
- Manager table features

Device New Folder and New

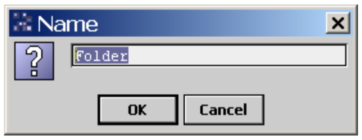
Buttons New Folder and New exist in the Device Manager view of a device-level component under *any* driver network. Use of New Folder is optional. Typically, you use New only if the driver does not support online discovery, *or* if you are programming offline.

Note: *New Folder and New are also on the Device Manager toolbar and the Manager menu.*

New Folder

The **`New Folder`** button in the Device Manager adds a special “DeviceFolder” component you can use to organize devices. This is equivalent to copying that same component from that driver’s palette. A Name dialog lets you name the folder before it is created (Figure 1-18).

Figure 1-18 *Example New Folder dialog in Device Manager*



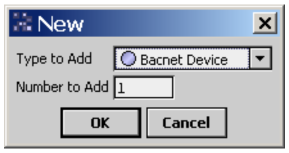
When you click OK, a new DeviceFolder component is added under the network.

Note: *A driver’s DeviceFolder component is different than a “normal” Folder component, as it provides that driver’s Device Manager view by default (just like the parent network component). Also, an “*`All Descendants`*” command is available from the root Device Manager, which allows you to see all devices in the network. For details, see “All Descendants” on page 1-16.*

New

The New button in the Device Manager allows you to add a new device component to the station. This is equivalent to copying a device component from that driver's palette. The New dialog provides a selection list for device type, and also the number of devices to add (Figure 1-19).

Figure 1-19 *Example New dialog in Device Manager*



Type to Add: Bacnet Device
Number to Add: 1

When you click OK, a new device-level component is added under the network. You usually need to edit specific details for each new device, including addressing parameters for communications.

All Descendants

When in the Device Manager of a device folder, it 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 selecting it from the Manager menu (Figure 1-20) or by simply clicking the "All Descendants" tool on the toolbar (Figure 1-21).

Note: *Be aware that in a large network, with many device folders and devices, using the all descendants feature may decrease performance. This might result from so many devices/points being subscribed.*

Figure 1-20 *Manager menu "All Descendants" command*

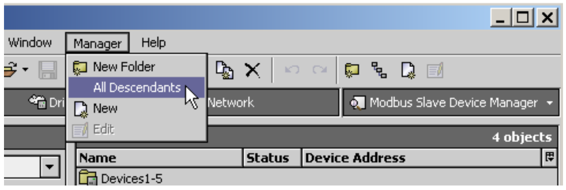
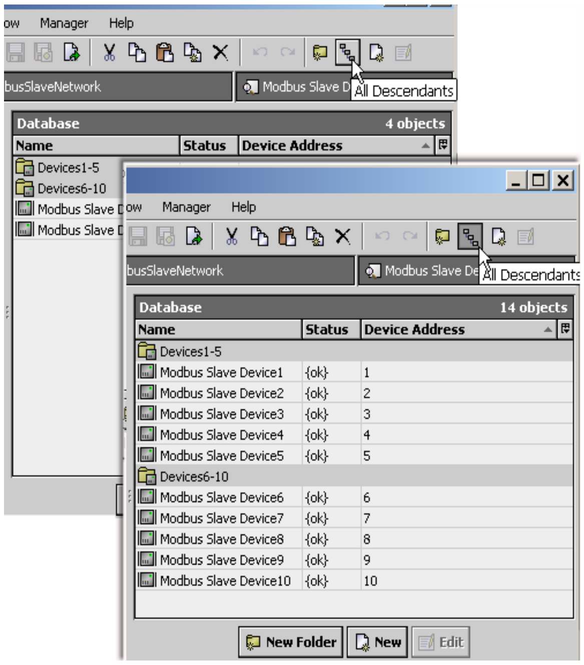


Figure 1-21 *Device Manager "All Descendants" tool on toolbar*



Database — 14 objects

Name	Status	Device Address
Devices1-5		
Modbus Slave Device1	{ok}	1
Modbus Slave Device2	{ok}	2
Modbus Slave Device3	{ok}	3
Modbus Slave Device4	{ok}	4
Modbus Slave Device5	{ok}	5
Devices6-10		
Modbus Slave Device6	{ok}	6
Modbus Slave Device7	{ok}	7
Modbus Slave Device8	{ok}	8
Modbus Slave Device9	{ok}	9
Modbus Slave Device10	{ok}	10

Note: *If you are using device folders, and you click on a table column to resort devices, please be aware that any device-to-device folder organization is lost during that view. However, you can always see contents of device folders clearly in the Nav tree, and again when returning to the Device Manager view from another view.*

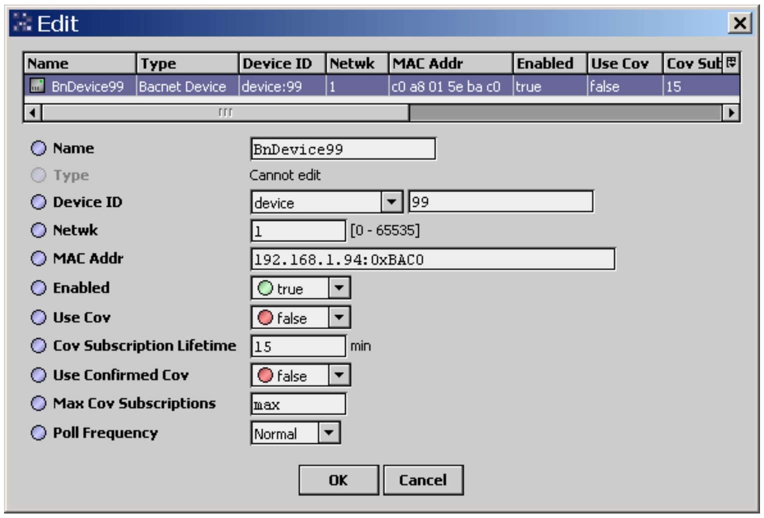
Device Edit

In the Device Manager view, you can edit any device component in the station database by simply double-clicking it. (Editing does not apply to device folders.)

Edit

The Edit dialog appears with the device listed (Figure 1-22).

Figure 1-22 *Example Edit dialog in Device Manager (single device)*



Edit

Name	Type	Device ID	Netwk	MAC Addr	Enabled	Use Cov	Cov Sub
BnDevice99	Bacnet Device	device:99	1	c0 a8 01 5e ba c0	true	false	15

Name	BnDevice99
Type	Cannot edit
Device ID	device 99
Netwk	1 [0 - 65535]
MAC Addr	192.168.1.94:0xBAC0
Enabled	true
Use Cov	false
Cov Subscription Lifetime	15 min
Use Confirmed Cov	false
Max Cov Subscriptions	max
Poll Frequency	Normal

Note: *The Edit dialog for a device component only shows some of the device component's properties. Typically, these are key properties required for communications, and will vary among drivers (to access all properties of the device component, go to its property sheet).*

When a single device is selected in the Edit dialog, you can edit any property except Type *(it was fixed when you added the device). Included is the ability to edit the Name of the device in the station. This is equivalent to the right-click Rename command on the component.*

The following related topics also apply:

- Device "gang" edits
- Manager table features

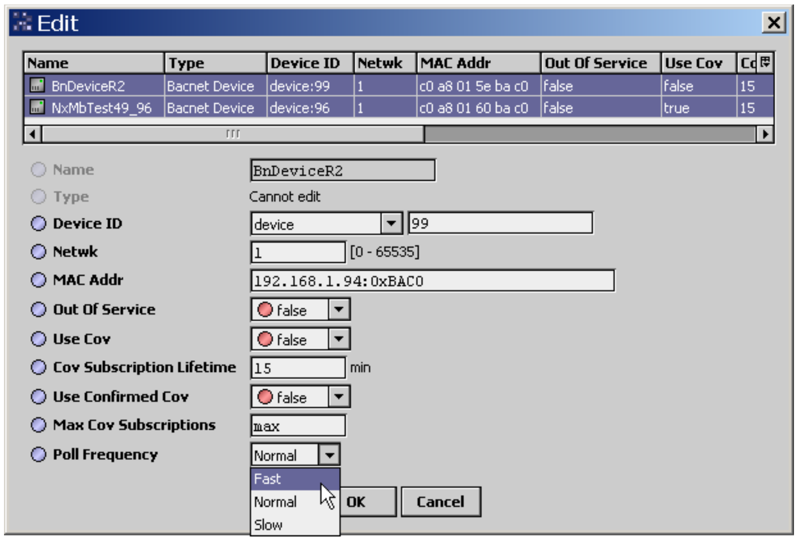
Device "gang" edits

The Edit button in the Device Manager allows you to edit one or more device components in the station in a single dialog. Before clicking **`Edit`**, use standard Windows keyboard controls to highlight (select) *multiple devices* (e.g. hold down Ctrl key and click desired devices).

Note: *Edit is also available on the Device Manager toolbar and the Manager menu.*

The "gang" edit feature is useful for making the same change in multiple (selected) devices. For example, as shown in Figure 1-23, you can change "Poll Frequency" in multiple devices at the same time (versus editing this individually).

Figure 1-23 *Example "gang edit" in Edit dialog in Device Manager*


Edit

Name	Type	Device ID	Netwk	MAC Addr	Out Of Service	Use Cov	Cc
BnDeviceR2	Bacnet Device	device:99	1	c0 a8 01 5e ba c0	false	false	15
NxMbTest49_96	Bacnet Device	device:96	1	c0 a8 01 60 ba c0	false	true	15

Name	BnDeviceR2
Type	Cannot edit
Device ID	device 99
Netwk	1 [0 - 65535]
MAC Addr	192.168.1.94:0xBAC0
Out Of Service	false
Use Cov	false
Cov Subscription Lifetime	15 min
Use Confirmed Cov	false
Max Cov Subscriptions	max
Poll Frequency	Normal

When you have the Edit dialog open with multiple devices, you can also click on select ones to make individual edits (e.g. Name, or any other editable property), during the same dialog session where you made "gang" property edits. When you click OK, all the changes are applied to the device components as made during your Edit session. For more details, see "About gang edits."

Note: *When you have multiple devices selected in the Edit dialog, properties that must be unique (such as Name) are automatically unavailable (dimmed). However, note that some properties that typically should be unique (often address properties) may still be available for "gang edit," as this rule is not automatically enforced. Typically, when editing these properties, you should verify only a single device component (row) is highlighted in the table.*

About Device Discover, Add and Match (Learn Process)

Online "device learns" are possible using the Device Manager for many drivers, for example the NiagaraNetwork, BacnetNetwork, LonNetwork, and NdioNetwork. Whenever available, this method is the easiest way to accurately add device components to the station database.

Note: *With the exception of a "Quik Learn" option in a LonNetwork, any device learn in NiagaraAX is a two-step process where you first:*

1. Discover device *candidates* for inclusion in the station database.
2. Select and Add from those candidates, creating device components in the network.
   If you already added devices using New, you can also **`Match`** candidates to existing devices. For more details see "Match (Device)" on page 1-20.

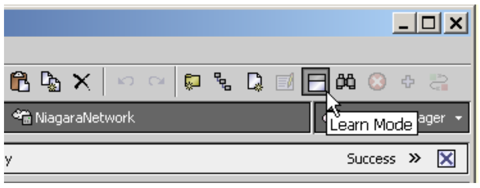
The Device Manager reinforces this process by providing two separate panes in the view whenever you enter "Learn Mode." See "About Learn toggle" on page 1-18.

About Learn toggle

Any driver that offers online "learns" of devices, points, and possibly schedules and log data (histories) provides specialized "Manager" views. Any such view offers both a two-pane view (Learn Mode) and a single-pane view (Database only).

At any time, you can *toggle* between Learn Mode and the single-pane (Database) view by clicking the Learn Mode tool in the toolbar (Figure 1-24), or using the Learn Mode command in the Manager menu. Also, note that the Discover tool (binoculars) is next to the Learn Mode tool.

Figure 1-24 *Learn Mode toggle tool in any Manager*

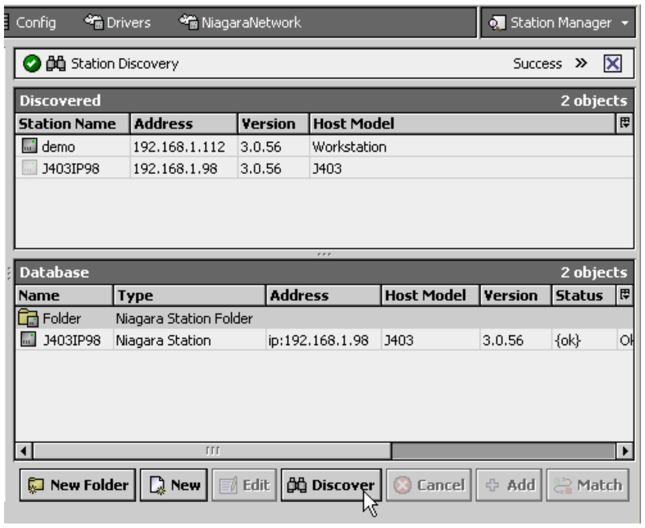


Note: *Whenever in Learn Mode of any Manager view (DeviceManager, PointManager, and so forth) you can drag the border between the two panes to resize, as necessary.*

Discover

The Discover button is available in the Device Manager only if the driver supports online discovery. When you click Discover, the Device Manager view splits into two panes (Learn Mode), and at the same time typically launches a discovery "job" (Figure 1-25).

Figure 1-25 *Discover splits Device Manager view*



Station Discovery Success

Discovered 2 objects

Station Name	Address	Version	Host Model
demo	192.168.1.112	3.0.56	Workstation
J403IP98	192.168.1.98	3.0.56	J403

Database 2 objects

Name	Type	Address	Host Model	Version	Status
Folder	Niagara Station Folder				
J403IP98	Niagara Station	ip:192.168.1.98	J403	3.0.56	{ok}

Note: *In some drivers, an intermediate dialog may appear before the discovery job begins. For example, in a discover for a BacnetNetwork, a "Configure Device Discovery" dialog allows you to limit possible ranges of devices, before discovery begins. For more details, see the Bacnet Users Guide.*

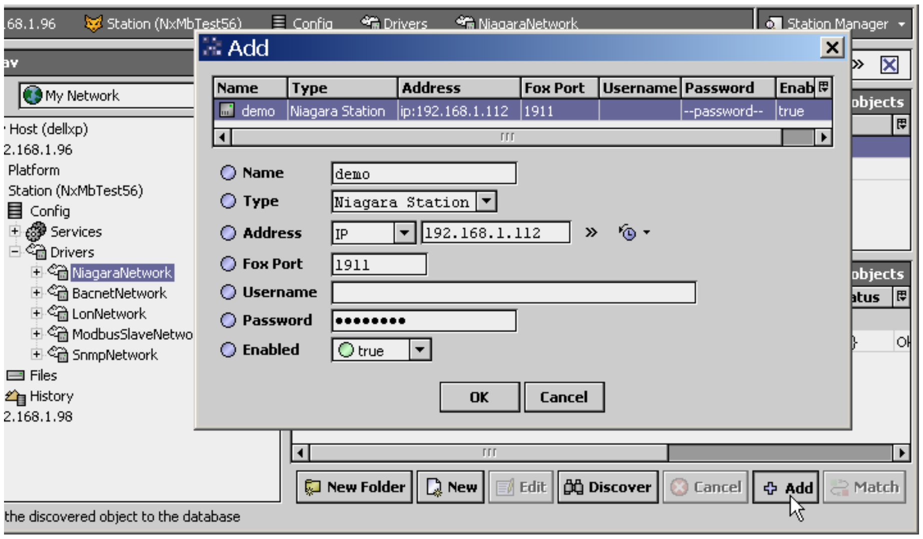
The two panes in the Device Manager operate as follows:

- **Discovered (top pane)**
  Lists devices discovered on the driver's network, as candidates. Any device found *that already exists* in the station database appears "ghosted" (faintly listed). A job "progress bar" is also included on top of the Discovered pane.
  Note: *A Cancel button is available during a discovery job. If needed, use it to stop discovery.*
- **Database (bottom pane)**
  Lists devices and device folders that are currently in the station database.

Add

The Add button is available in the Device Manager in Learn Mode when you have one or more devices selected (highlighted) in the top Discovered pane. When you click Add, an Add dialog appears that allows you to edit items before the device component(s) are created in the station database (Figure 1-26).

Figure 1-26 *Add dialog from Add button in Device Manager*



Add

Name	Type	Address	Fox Port	Username	Password	Enab
demo	Niagara Station	ip:192.168.1.112	1911		--password--	true

Name: demo
Type: Niagara Station
Address: IP 192.168.1.112
Fox Port: 1911
Username:
Password: ••••••••
Enabled: true

The Add dialog is nearly identical to the device Edit dialog, but allows you to edit Type as well as other device properties. Often, *device address* properties in the Add dialog already have acceptable values for operation (otherwise, communications to that device would not have occurred). Often, you change only Name, unless you know other settings you wish to change. You can always Edit the device component(s) after you click OK and add them to your station.

Note: *When you have one or more devices selected in the top Discovered pane, an Add tool is also available on the toolbar ("plus" symbol), as well as a command in the Manager menu. Also, you can simply double-click a discovered device to bring it up in the Add dialog.*

Match (Device)

Device **`Match`** is a feature that may be useful when you have an application replicated many times at the device level, or if you have programmed offline using the New device feature.

- In the first case (replicated device application), you could discover and add *one* typical device, and complete further engineering under it (learning and adding proxy points, point extensions, creating other control logic, adding Px views, including all *self-contained* links and bindings).
  Then, you could *duplicate* that typical device component (choosing Duplicate in its right-click menu) for as many identical devices as exist. The Match feature now allows you to *match* each duplicated device component to a *unique* discovered device, saving engineering time. This repopulates the necessary properties of the duplicated device object with the correct values from the discovered device.
- In the second case (offline programming) where a connection to the actual device network is unavailable, you can manually add New devices and begin station engineering of a driver network. Typically, most component creations under a driver network are possible (including all levels) using the New feature in the various "manager" views (Device Manager, Point Manager, other device extension managers). Or, you can add saved applications (from the device level on down) and edit as necessary. Then, when online with the driver network later, you could use Match to "sync" to existing components (device-level, proxy points, and so forth).

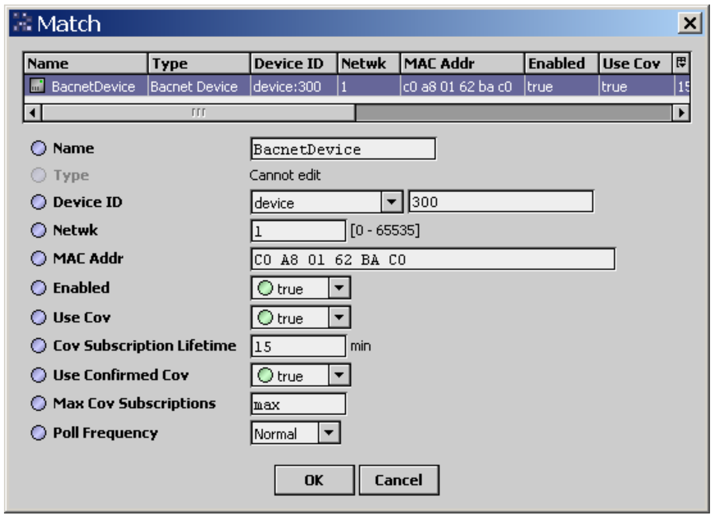
The Match button in the Device Manager becomes available when in Learn Mode, and you have:

1. Selected *one* device candidate in the *top* (Discovered) pane.
2. Selected *one* existing device component in the *bottom* (Database) pane.

Note: *In this case, the toolbar also has an available Match tool, and the Manager menu has a Match command.*

When you click Match, the Match dialog appears, as shown in Figure 1-27.

Figure 1-27 *Match dialog example in Device Manager*


Match

Name	Type	Device ID	Netwk	MAC Addr	Enabled	Use Cov
BacnetDevice	Bacnet Device	device:300	1	c0 a8 01 62 ba c0	true	true

Name	BacnetDevice
Type	Cannot edit
Device ID	device 300
Netwk	1 [0 - 65535]
MAC Addr	C0 A8 01 62 BA C0
Enabled	true
Use Cov	true
Cov Subscription Lifetime	15 min
Use Confirmed Cov	true
Max Cov Subscriptions	max
Poll Frequency	Normal

Note: *Match is strictly a "one-to-one" function for "discovered-to-database"—note that it is unavailable any time you have multiple items selected either in either the top or bottom pane.*

The Match dialog is nearly identical to the single-device Edit dialog, and typically provides the "discovered" device address values. Often, most properties in the Match dialog already have acceptable device *address* values required for operation (otherwise, communications to the discovered device would not have occurred). You can always Edit the device component after you click OK and add it to your station.

Manager table features

As with other table-based views in Workbench, manager views of various components under a driver's network provide common features that can be useful (see "About table controls" for a general overview). These features apply to various component views found throughout a station's driver architecture, for example, the Driver Manager, Device Manager, Point Manager, History Import Manager, and so on.

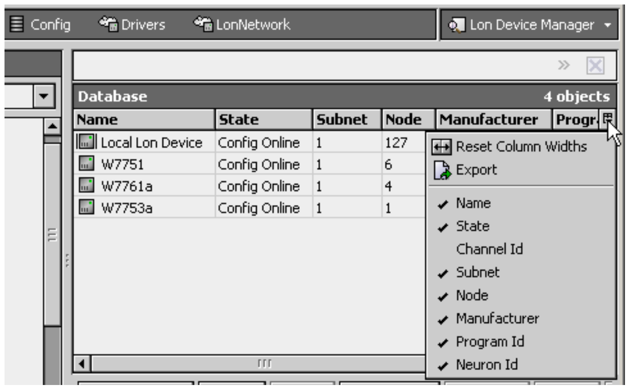
Of particular utility are the following table features:

- Table options menu
- Column resorting

Table options menu

Use the table options drop-down menu (small control in upper table right, as shown in Figure 1-28) to perform any of the following:

Figure 1-28 *Table options menu in Device Manager*



Database 4 objects

Name	State	Subnet	Node	Manufacturer	Progr
Local Lon Device	Config Online	1	127		
W7751	Config Online	1	6		
W7761a	Config Online	1	4		
W7753a	Config Online	1	1		

- **Reset Column Widths**
  Useful if you manually changed widths of columns, and now some contents are hidden (even after scrolling). The reset restores all columns to widths that accommodate contents.
- **Export**
  Produce the standard Export dialog, where you can select exporting the table to PDF, text, HTML, or CSV (comma separated variable). See "About table exports" for more details.
- **select or clear columns for display**
  Depending on the driver, a certain "default" collection of columns is pre-selected for display in the Manager view. You can change that by checking or clearing column headings.

Column resorting

Click on any column header to toggle the table sort between ascending (first click) and descending (second click). The current table sort is indicated by a small triangle in the sorting column.

Common device components

Each device component has some number of required (frozen) slots and properties, which will vary by driver. However, the following categories of components and slots are typical to most device-level components:

- Device status properties
- Device Alarm Source Info
- Device address properties
- Driver-specific device slots

Also, a device component has one or more device *extensions*, which are visible when you expand the device in the Nav tree. For more details, see "Types of device extensions" on page 1-26.

Note: *A few drivers implement "virtual components," which typically adds a frozen "Virtual" gateway slot under the device component. See "Virtual gateway and components" on page 1-23.*

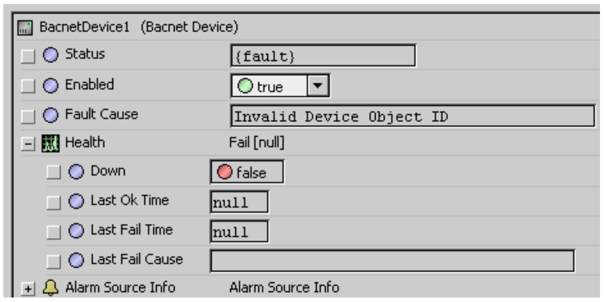
Device status properties

Status

The Status property of a device indicates whether it is communicating—good status is "{ok}" (no status flags). However, if using New to add a device, initial Status may be down or fault, as shown in Figure 1-29. This could mean device address properties are yet unassigned, or are misconfigured.

Note: *If a device status fault, see the* Fault Cause *property value for more details.*

Figure 1-29 *Device status properties*



BacnetDevice1 (Bacnet Device)	
Status	{fault}
Enabled	true
Fault Cause	Invalid Device Object ID
Health	Fail [null]
Down	false
Last Ok Time	null
Last Fail Time	null
Last Fail Cause	
Alarm Source Info	Alarm Source Info

Status of devices is verified by successful polling, or the device status "ping" as configured in the network component's Monitor configuration. See "About Monitor" on page 1-7.

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

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

Enabled

By default, device Enabled is true—you can toggle this in the property sheet, or in the Device Manager (by selecting the device and using the Edit button). See Caution on page 5.

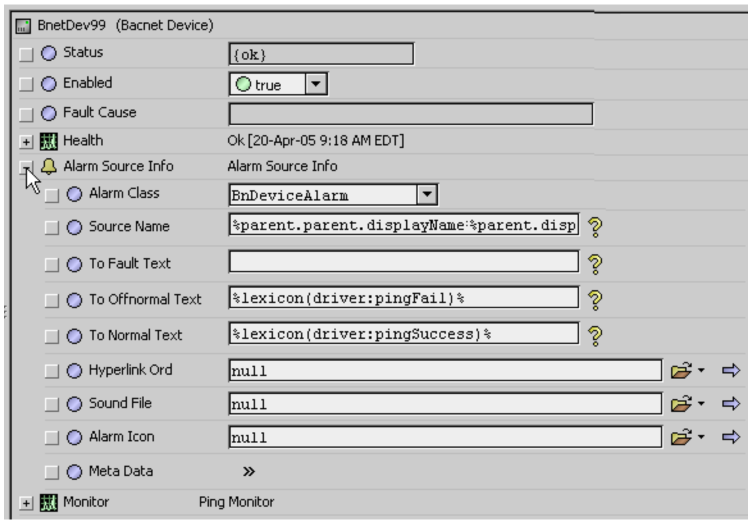
Health

Device Health contains historical properties about the last successful message received from the device, including timestamps.

Device Alarm Source Info

A device's Alarm Source Info slot holds a number of common alarm properties (Figure 1-10). These properties are used to populate an alarm if the *device* does not respond to a monitor ping. This ping is configured at the network level—see "About Monitor" on page 1-7.

Figure 1-30 *Example Device Alarm Source Info properties*



BnetDev99 (Bacnet Device)	
Status	{ok}
Enabled	true
Fault Cause	
Health	Ok [20-Apr-05 9:18 AM EDT]
Alarm Source Info	Alarm Source Info
Alarm Class	BnDeviceAlarm
Source Name	%parent.parent.displayName%parent.disp
To Fault Text	
To Offnormal Text	%lexicon(driver:pingFail)%
To Normal Text	%lexicon(driver:pingSuccess)%
Hyperlink Ord	null
Sound File	null
Alarm Icon	null
Meta Data	»
Monitor	Ping Monitor

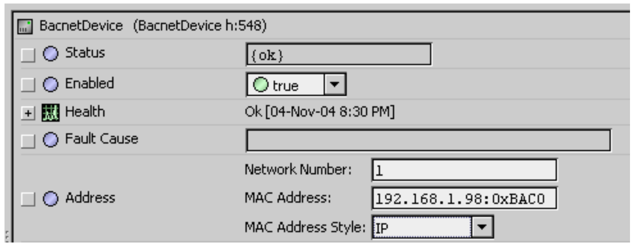
These properties work the same as those in an alarm extension for a control point. For property descriptions, see the *User Guide* section "About alarm extension properties".

Note: *Each parent network component also has its own Alarm Source Info slot, with identical (but independently maintained) properties. See "About network Alarm Source Info" on page 1-6.*

Device address properties

Depending on the driver, a device-level component typically has one or more address type properties, accessible from the device's property sheet. For example, a BacnetDevice has an Address property with 3 separate fields (Figure 1-31).

Figure 1-31 *Example device address properties*



BacnetDevice (BacnetDevice h:548)
Status {ok}
Enabled true
Health Ok [04-Nov-04 8:30 PM]
Fault Cause
Address
Network Number: 1
MAC Address: 192.168.1.98:0xBAC0
MAC Address Style: IP

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

Poll Frequency

Common to most devices is a Poll Frequency property, typically located *below* device address properties. It provides a drop-down menu to select among 3 poll frequencies, as configured in the network's Poll Service. For more details, see "About poll components" on page 1-10.

Driver-specific device slots

A device-level component typically has slots unique to that driver, accessed on its property sheet. For example, a Lon device has a large collection of nv slots (network variables nvi, nvo, and nci), each as an expandable container holding numerous values. A BacnetDevice holds a number of properties that define Bacnet "services" that it supports. For details, see the specific driver document.

Virtual gateway and components

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 (See "About device components" on page 1-3.) In the NiagaraAX driver architecture, a device's virtual gateway provides access to "virtual components" in the station's "virtual component space," specific to that device.

At the time of this document update, there are *two* drivers that have implemented virtual components: the BACnet driver, and the AX-3.4 and later niagaraDriver (Niagara Network). Details on the latter are in this document, see "About Niagara virtual components" on page 2-38.

Virtual components are sometimes referred to as "virtual points", quite different from the "proxy point" components found in most drivers. The term "virtual point" is a actually a misnomer to describe the niagaraDriver implementation, as Niagara virtual components reflect whatever component types are in the source remote NiagaraAX station (not just control points).

Note: *Baja virtual components are not specific to just drivers. However, the original need for virtual components (which are transient, and do not permanently reside in a station's database) and a persisted "gateway" component to access them, came about from driver "use case needs." It is possible that other applications for virtual gateways (with virtual components) may evolve in future builds of NiagaraAX.*

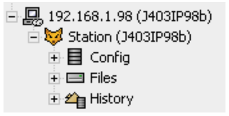
See the following sections for more details:

- About virtual component spaces
- About virtual gateways
- Application and limitations of virtual components
- Virtual components in Px views
- Virtual ord syntax

About virtual component spaces

A NiagaraAX station contains object "types," each within a defined "space" at a top level (Figure 1-32).

Figure 1-32 *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.

A station uses these objects when running, and when it is stopped they persist—components in the station's database (config.bog), files in the station's directory, and histories within the history database.

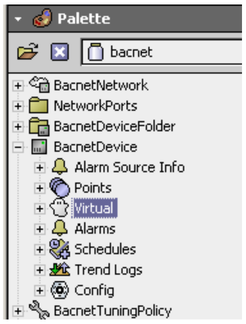
Virtual component spaces are *different*, in the following ways:

- There can be *multiple* virtual component spaces—each belongs to a different virtual gateway, which are specialized 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 are accessed, that is become subscribed.
- Virtual components are transient components created in the station only when needed. When no longer necessary (i.e. become unsubscribed) they are automatically removed in 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). For example, links to and from virtual components, use of point extensions (history, alarm, etc.), and typical copy/paste operations are not supported.

About virtual gateways

Virtual gateways reside 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. In the initial "driver application" of virtual components, each *device* component in a driver network has its own single virtual gateway, as shown in Figure 1-33 for a BacnetDevice in the `bacnet` palette.

Figure 1-33 *One virtual gateway per device*



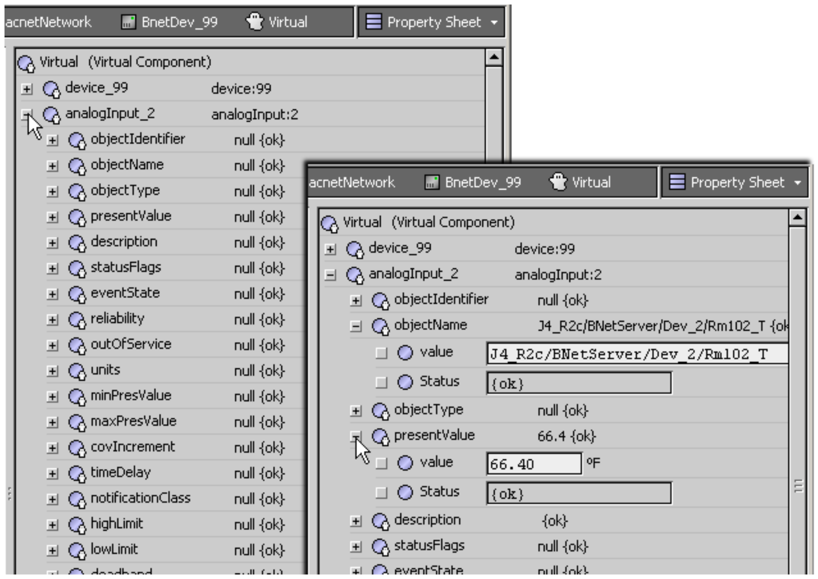
This is a logical division of gateways (separating data by its source device). However, note that future virtual component applications may not always use this "one-to-one-device" hierarchy for virtual gateways. At some future point, a driver may have a "network level" virtual gateway, or possibly provide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

Gateway activation

Unlike with most device *extensions*, there is no special view for a virtual gateway—in Workbench you can simply double-click it to access the gateway's property sheet, or expand it in the Nav tree. When you do this for a device's virtual gateway, a "driver-specific" call is made to the device to gather data, where the results appear as child *virtual components*.

In the case of the BACnet driver, expanding a virtual gateway fetches "an object list," where each BACnet object in the device appears as a virtual component (slot) under the gateway. You can further expand each property of an object to see its "virtual properties," including a value and poll status. See Figure 1-34.

Figure 1-34 *Virtual properties of BacnetVirtualComponent*



```
Virtual  (Virtual Component)
  device_99              device:99
  analogInput_2          analogInput:2
    objectIdentifier       null {ok}
    objectName             null {ok}
    objectType             null {ok}
    presentValue           null {ok}
    description            null {ok}
    statusFlags            null {ok}
    eventState             null {ok}
    reliability            null {ok}
    outOfService           null {ok}
    units                  null {ok}
    minPresValue           null {ok}
    maxPresValue           null {ok}
    covIncrement           null {ok}
    timeDelay              null {ok}
    notificationClass      null {ok}
    highLimit              null {ok}
    lowLimit               null {ok}
```

```
Virtual  (Virtual Component)
  device_99              device:99
  analogInput_2          analogInput:2
    objectIdentifier       null {ok}
    objectName             J4_R2c/BNetServer/Dev_2/Rm102_T {ok
      value                J4_R2c/BNetServer/Dev_2/Rm102_T
      Status               {ok}
    objectType             null {ok}
    presentValue           66.4 {ok}
      value                66.40  ºF
      Status               {ok}
    description            {ok}
    statusFlags            null {ok}
```

In this case, each "virtual property" component is a virtual point. For further details about the BACnet implementation, see the "About Bacnet virtual points" section in the *NiagaraAX BACnet Guide*. For details about virtual components in the NiagaraNetwork of a Supervisor, see "About Niagara virtual components" on page 2-38.

Application and limitations of virtual components

In a NiagaraAX driver that offers virtual components, there are two main applications for "virtual points":

- Px view bindings for simple polling of values, without the station resource overhead of (persisted) proxy points. Upon being unsubscribed, virtual components are simply removed from the driver's poll scheduler (or subscription mechanism), and also station memory. The only persisted parts are the *ords* (using virtual syntax) in the Px widget bindings. In some cases, particularly with replicated device applications, this allows a JACE station to graphically monitor *many* more devices than if using just proxy points. See the next section "Virtual components in Px views" for more details.
- Quick review and (if possible) adjustments to one or more properties in objects in native devices, from the Workbench property sheets of virtual components. Otherwise, you might need to create proxy points, then delete them after reviewing and changing properties. This application applies to both Niagara driver virtual components as well as to BACnet virtual components.

As for limitations, note that virtuals are *transient* vs. persisted components—they are dynamically created (and subscribed) only when accessed, and 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 under virtual components. These things require use of proxy points, which are persisted in the station database.

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

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

Note that often proxy points are used for monitoring only, becoming subscribed only when a user is looking at a Px view, then becoming unsubscribed when not being viewed. However, such proxy points persist in the station database always—consuming station resources. The difference with using virtual component in this application is that they not only become unsubscribed, but are *removed* from the station's memory. The only persisted part is the "ord" to the virtual components in the Px widget bindings.

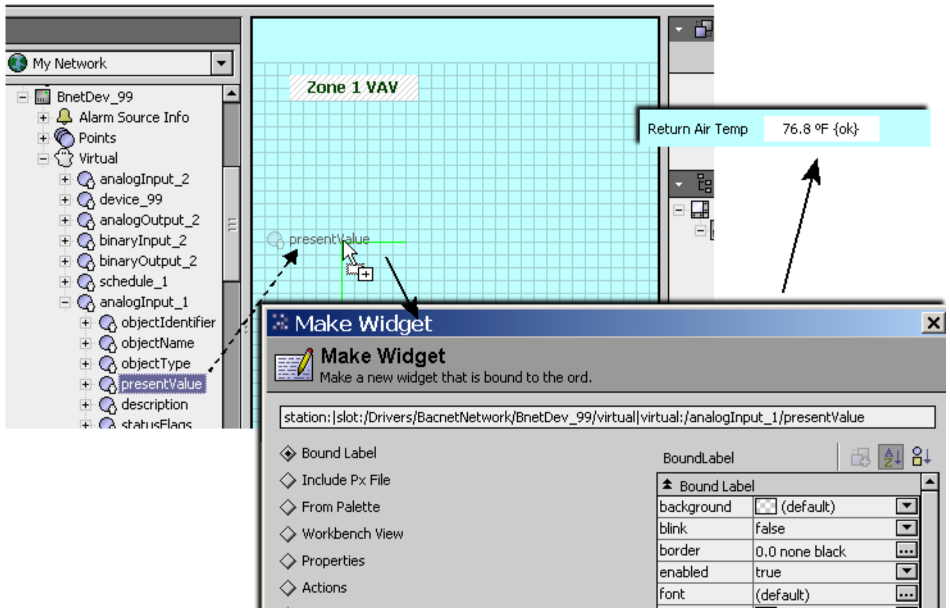
Virtual components in Px views

As previously mentioned, virtual components can be used to show real-time values in Px views—at least when all the rich features of proxy points are not required.

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 only persisted (permanent) record of such a virtual component is its *ord* in the Px binding to it, which uses a "Virtual ord syntax" that includes the virtual gateway within it (and activates it at runtime). This ord is automatically resolved when you drag a virtual component onto the Px page, and make your selection in the popup Px "**`Make Widget`**" dialog, as shown in the Bacnet example in Figure 1-35.

Figure 1-35 *Dragging Bacnet virtual component into Px editor view, with resulting Make Widget popup*



station:|slot:/Drivers/BacnetNetwork/BnetDev_99/virtual|virtual:/analogInput_1/presentValue

background	(default)
blink	false
border	0.0 none black
enabled	true
font	(default)

Depending on driver, Px usage details may differ for virtual components. For example, usage of Niagara virtual components in Px views includes special considerations. For details, see "Niagara virtuals in Px views" on page 2-47.

For complete details on Px views and widget editing, see the "About Px Editor" section in the *User Guide*.

Virtual ord syntax

The general syntax for a "virtual ord" uses a specialized form as follows:

`<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.

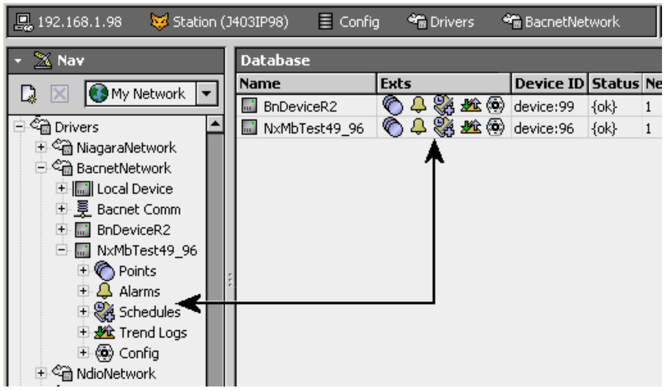
Note: *Starting in AX-3.7, virtual components in a NiagaraNetwork (Niagara virtuals) use a much simpler virtual ord than in previous releases, due to a new "virtual cache" mechanism. Much information that was formerly in the "virtualPath" portion of the ord syntax now resides in this cache. For related details, see "Niagara virtuals in AX-3.7" on page 2-38 and "Ords for Niagara virtual components" on page 2-42.*

Types of device extensions

When you create a device-level component (New, Add, or simply drop from the driver's palette), you may notice that it has default *children*, collectively called *device extensions*. Device extensions group various functions of a device.

Note: *For any device, its extensions are typically visible both in the Nav tree and in the Device Manager view (Figure 1-36), providing double-click access to each extension's default view. One exception to this is for a Supervisor, in its Station Manager view of the NiagaraNetwork, where special "provisioning" extensions for NiagaraStations do not appear. See "NiagaraStation component notes" on page 2-14 for related details.*

Figure 1-36 *Device extensions in Nav tree and Device Manager*



Name	Exts	Device ID	Status	Ne
BnDeviceR2		device:99	{ok}	1
NxMbTest49_96		device:96	{ok}	1

Perhaps the most important of all device extensions is the *Points* extension—the container for all proxy points (representing real-time data originating from that device).

Common types of Device extensions include:

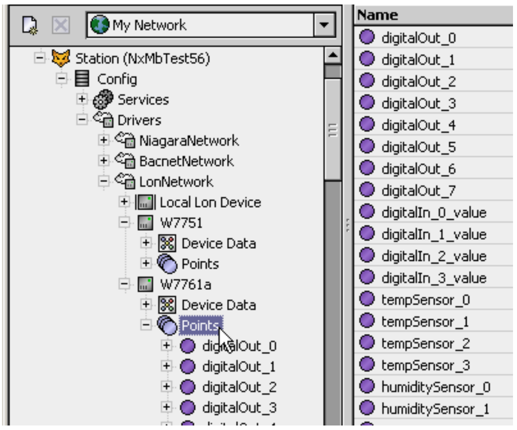
- Points extension—see "About the Points extension" on page 1-27.
- Histories extension—see "About the Histories extension" on page 1-34.
- Alarms extension—see "About the Alarms extension" on page 1-35.
- Schedule extension—see "About the Schedules extension" on page 1-36.

Note: *The* **`NiagaraStation`** *component (device in a NiagaraNetwork) has* additional *device extensions, including a "***`Users`***" extension, and starting in AX-3.5, a "***`Files`***" extension and "***`Sys Def`***" extension. For more details, see "NiagaraStation component notes" on page 2-14.*

About the Points extension

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

Figure 1-37 *Points under a Lon device*



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

Note: *You create all proxy points using the Point Manager view of Points—simply double-click Points under any device component, or right-click Points and select* **`Views > Point Manager`**. *See "About the Point Manager" on page 1-37 for more details.*

For general information on control points, refer to "About control points" in the *User Guide*. See the next section "About proxy points" for further details on proxy points.

About proxy points

Proxy points are often the bulk of all points in a station. This is true whether a JACE or a Supervisor station. Proxy points *are* any of the 8 simple control points (BooleanPoint, BooleanWritable, EnumPoint, EnumWritable, and so forth), only *with a non-null proxy extension* (for general control point information, refer to the *User Guide* section "About the proxy extension".)

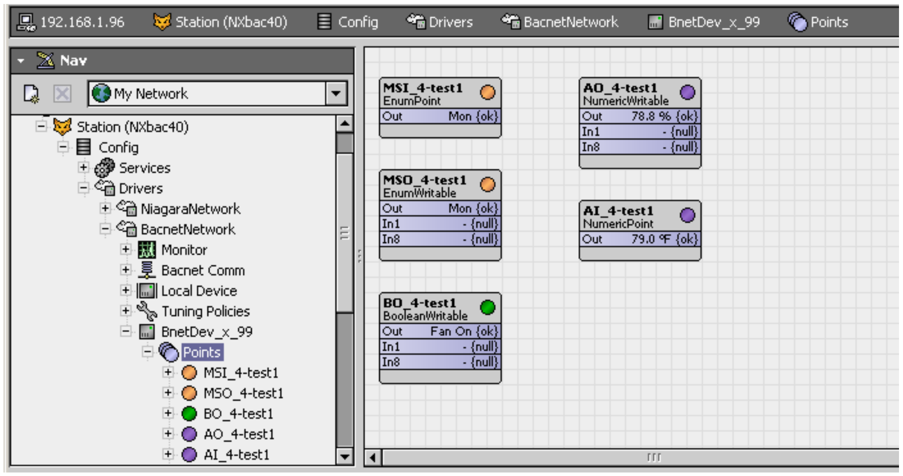
These following sections provide more details about proxy points:

- Location of proxy points
- How proxy points are made
- Proxy points versus simple control points
- ProxyExt properties

Location of proxy points

In the AX station architecture, proxy points must reside under the driver network and device from which the specific data originates. Proxy points are under that device's Points container (Points extension), as shown in Figure 1-38.

Figure 1-38 *Proxy points location example*



As needed, you can create folders under a device's Points container to further organize the proxy points. In addition to proxy points, you can also add simple control points (null proxy ext), schedule objects, and other kitControl objects under a device's Points container.

Note: *See also "Location for kitControl components" in the* kitControl Guide*.*

Depending on station type, proxy point locations will vary as follows:

- JACE **station**
  Most proxy points are under non-Niagara driver networks, such as Lonworks, Bacnet, and Modbus, to name a few. These points represent real-time data in various devices attached (or somehow networked) to the JACE controller.
  Like all AX stations, a JACE typically has a Niagara Network too. Any proxy points under it will represent data received from other JACE stations and/or perhaps the Supervisor station.
- **Supervisor station**
  Typically, most proxy points are under its Niagara Network, where each JACE appears as a station device. Proxy points under each station device represent real-time data from that JACE. Typically, most data in a JACE station also originates as a proxy point, and so these points are often a "proxy of a proxy." For details, see "About the Niagara Network" on page 2-1.
  Note: *If the Supervisor is specially licensed for direct field device communications, such as a BACnet Supervisor or OPC Supervisor, its station will have many proxy points under other driver network types, and perhaps only a few under its NiagaraNetwork. In Drivers architecture, it resembles a JACE.*

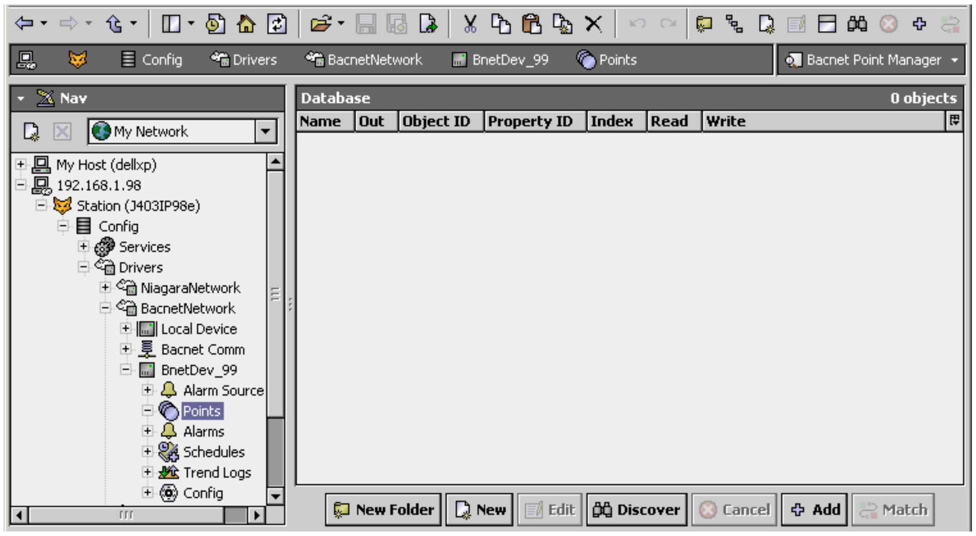
For more details, see "About Network architecture" on page 1-2.

How proxy points are made

Note: *You create proxy points in a station as one of the later stages in building a driver network. This section only provides an overview of the proxy points portion.*

When you add a device under a network, one of its default extensions is a container named "Points." The default view for Points is a "Point Manager," which you use to add proxy points to the station database. An example Bacnet device Points Manager is shown in Figure 1-39.

Figure 1-39 *Points Manager example (Bacnet Device)*

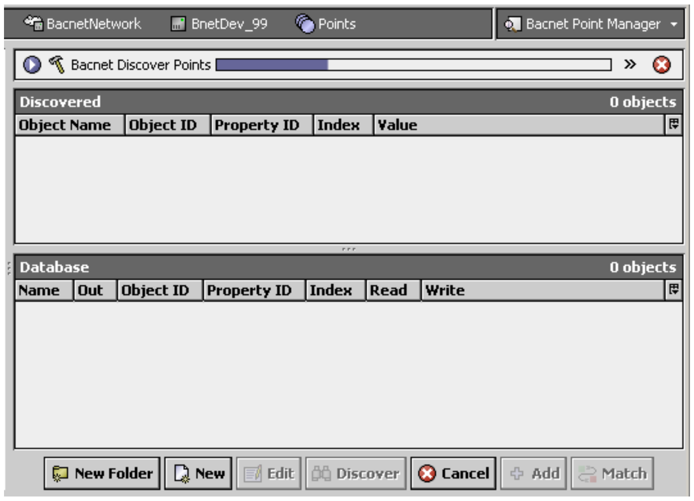


Typically, you have performed your previous station configuration of this network with the host (e.g. JACE) networked and communicating to the devices of interest. This allows you to use the “Learn Mode” feature (provided by most drivers). This feature is especially useful for adding proxy points to the station.

In the Point Manager, the Learn Mode toggles the view between only proxy points that are currently in the station (“Database”), and a split view showing both a “Discovered” area and the “Database” area. See “About Learn toggle” on page 1-18 for more details.

Clicking the `Discover` button launches a “point discovery job.” The driver queries the selected device to retrieve available data. Depending on a number of factors (driver type, communications rate, amount of data), this can take from only a few seconds to over a minute. See Figure 1-40.

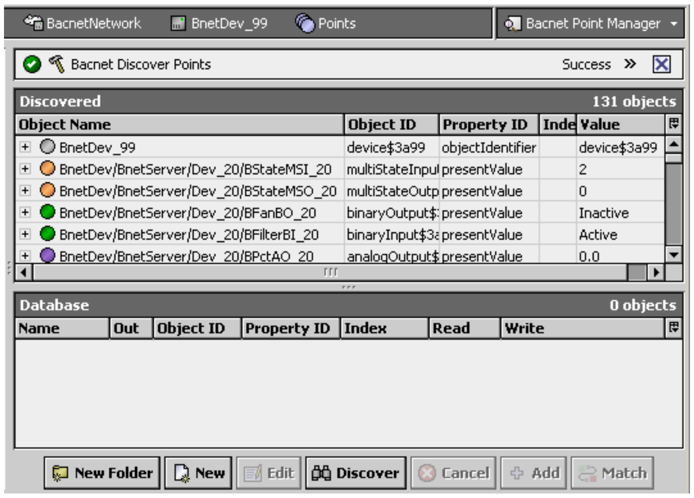
Figure 1-40 *Points Discover in progress*



When the discover completes, the “Discovered” view portion provides a table of available data items. Each row represents at least one item (a *candidate* for one proxy point). If there are multiple, closely-related data items, that row appears with a leading plus (“+”). You can expand it to see other available data items. Again, each row is a candidate for *one* proxy point.

Depending on the driver type, table column headings vary—for example a Bacnet points discover shows “Object Name,” “Object ID,” etc., in the Discovered table (see Figure 1-41).

Figure 1-41 *Points Discover completed*



Discovered 131 objects

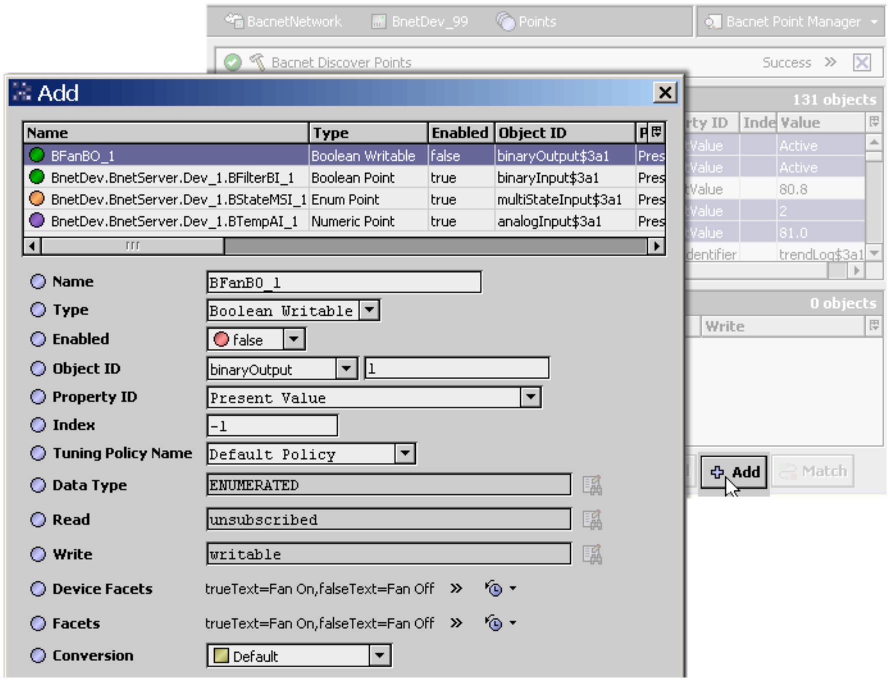
Object Name	Object ID	Property ID	Inde	Value
BnetDev_99	device$3a99	objectIdentifier		device$3a99
BnetDev/BnetServer/Dev_20/BStateMSI_20	multiStateInpul	presentValue		2
BnetDev/BnetServer/Dev_20/BStateMSO_20	multiStateOutp	presentValue		0
BnetDev/BnetServer/Dev_20/BFanBO_20	binaryOutput$:	presentValue		Inactive
BnetDev/BnetServer/Dev_20/BFilterBI_20	binaryInput$3a	presentValue		Active
BnetDev/BnetServer/Dev_20/BPctAO_20	analogOutput$	presentValue		0.0

Database 0 objects

Name	Out	Object ID	Property ID	Index	Read	Write

As needed, you click on column headers to resort and/or use the scroll bar to view available discovered data items. After selecting one or more items by clicking on rows (to highlight), you can click the **`Add`** button to start the proxy point creation process. The Add dialog appears (Figure 1-42).

Figure 1-42 *Add points dialog*



Add

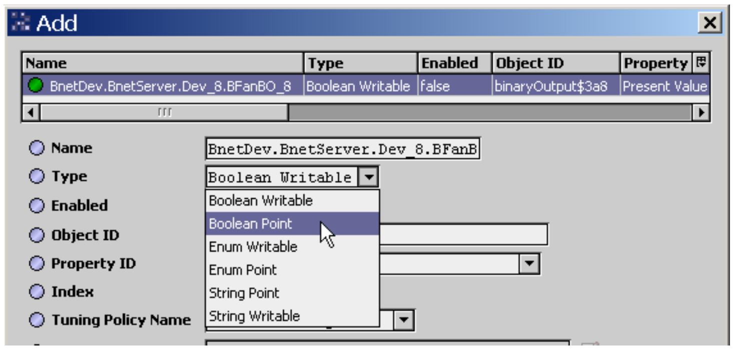
Name	Type	Enabled	Object ID	P
BFanBO_1	Boolean Writable	false	binaryOutput$3a1	Pres
BnetDev.BnetServer.Dev_1.BFilterBI_1	Boolean Point	true	binaryInput$3a1	Pres
BnetDev.BnetServer.Dev_1.BStateMSI_1	Enum Point	true	multiStateInput$3a1	Pres
BnetDev.BnetServer.Dev_1.BTempAI_1	Numeric Point	true	analogInput$3a1	Pres

Name	BFanBO_1
Type	Boolean Writable
Enabled	false
Object ID	binaryOutput 1
Property ID	Present Value
Index	-1
Tuning Policy Name	Default Policy
Data Type	ENUMERATED
Read	unsubscribed
Write	writable
Device Facets	trueText=Fan On,falseText=Fan Off
Facets	trueText=Fan On,falseText=Fan Off
Conversion	Default

The Add dialog allows you to select each data item and change things about its default point creation *before* it is added to the station database. Most important is “Type,” meaning the control point type—as unlike other things (such as Name) you cannot change that after the creation. Apart from Name and Type, most other properties are *proxy extension* properties.

Selecting the Type drop-down, alternate point types are available for selection, see Figure 1-43. If the driver recognizes the data item as writable, this will include writable point types.

Figure 1-43 *Type drop-down in Add dialog*



Name	Type	Enabled	Object ID	Property
BnetDev.BnetServer.Dev_8.BFanBO_8	Boolean Writable	false	binaryOutput$3a8	Present Value

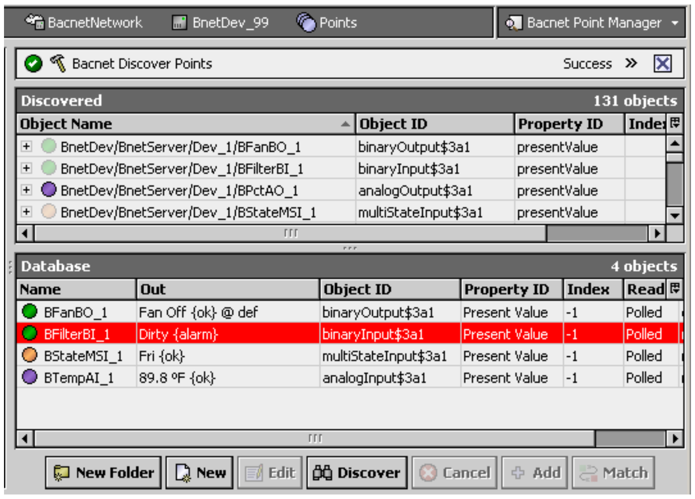
Name	BnetDev.BnetServer.Dev_8.BFanB
Type	Boolean Writable
Enabled	
Object ID	
Property ID	
Index	
Tuning Policy Name	

Typically, you do not change the data category type (Boolean, Numeric, etc.), but you may wish to select either a read-only point or a writable point.

You click any discovered data item to select it for changing Type, or any other property. If a common change is needed among data items (for example, Poll Frequency), you can select multiple items and edit that property one time.

When you are satisfied with the point types shown for each item, you click the OK button at the bottom of the Add dialog. Those proxy points are then created in the station database in the Points container, and now appear as table rows in the “Database” (lower) table of the Point Manager view. The source data items now appear “ghosted” in the “Discovered” (upper) table, to indicate that they now exist in the station database. See Figure 1-44.

Figure 1-44 *Proxy points added to station*



Discovered 131 objects

Object Name	Object ID	Property ID	Index
BnetDev/BnetServer/Dev_1/BFanBO_1	binaryOutput$3a1	presentValue	
BnetDev/BnetServer/Dev_1/BFilterBI_1	binaryInput$3a1	presentValue	
BnetDev/BnetServer/Dev_1/BPctAO_1	analogOutput$3a1	presentValue	
BnetDev/BnetServer/Dev_1/BStateMSI_1	multiStateInput$3a1	presentValue	

Database 4 objects

Name	Out	Object ID	Property ID	Index	Read
BFanBO_1	Fan Off {ok} @ def	binaryOutput$3a1	Present Value	-1	Polled
BFilterBI_1	Dirty {alarm}	binaryInput$3a1	Present Value	-1	Polled
BStateMSI_1	Fri {ok}	multiStateInput$3a1	Present Value	-1	Polled
BTempAI_1	89.8 °F {ok}	analogInput$3a1	Present Value	-1	Polled

You may repeat this process as needed to create additional proxy points, where you can toggle the display of the Discovered portion off and on by clicking the Learn Mode tool.

Once in the Database table of the Points Manager, you can click to select (highlight) a proxy point, then modify it using the **`Edit`** button, or simply double-click it. This produces an **`Edit`** dialog nearly identical to the **`Add`** dialog, where you can change the same proxy extension properties and Name (but *not* Type). You can also select *multiple* proxy points and use the **`Edit`** dialog to “gang edit” one or more of the same properties that are common to each selected point.

In the Database portion, you can right-click a proxy point to access its normal views, including property sheet. There, you can expand its Proxy Ext to see many of the same properties you saw in the Add and Edit dialogs. Also, you can right-click a proxy point in the Database table to access any available point actions (if a writable point).

Each Points container has other views besides the default Point Manager. For example, you can select its Wire Sheet view to see and organize the proxy points glyphs (Figure 1-38 on page 28), add additional objects from palettes control and/or kitControl, and so forth.

Note: *The following notes apply when working with proxy points.*

- In a Niagara Network, the Discover and Add process is different than in other driver networks. There, you use a “BQL Query Builder” to select data items in a remote station. For more details, see

"About the Niagara Network" on page 2-1 and "About the Bql Query Builder" on page 2-21.

- Any Point Manager view only shows *proxy* points. If you added other objects (for example, control points with null proxy extension, or kitControl objects), they do not appear in the Database table. However, they are visible in the Nav tree and the Points wire sheet.
- If you want *folders* under Points to organize your proxy points and other objects, use the "New Folder" button in the Point Manager to create each folder. This provides a Point Manager view for each folder. For more details, see "About the Point Manager" on page 1-37.

Proxy points versus simple control points

Functionally, there is little difference between a simple control point (NullProxyExt) and the equivalent proxy point. For example, you can add the same extensions (e.g. control, alarm, and history) to a proxy point as to a simple control point—there is no need to "duplicate" the point first.

However, apart from the location differences (see "Location of proxy points" on page 1-28) and manner of creation (see "How proxy points are made" on page 1-28), proxy points have the following differences from simple control points:

- **Status flag processing**
  Status flags of proxy points are affected by real-time changes that may occur in the remote device, plus changes in communications between that device and the station. This is in addition to "AX-added" status flags set by an alarm extension (for example, "alarm" or "unackedAlarm"). See "Proxy point status" on page 1-33.
  Related are "global" status rules common among NiagaraAX drivers, set at both the network-level as well as adjustable at the proxy-point level. For details, see "About Tuning Policies" on page 1-8.
- **Point duplication**
  When you duplicate a proxy point, you are duplicating information in its Proxy Ext that might be better left unique. This may result in redundant messages between the device and station for the same data item, adding overhead. If duplicating a proxy point, you should have a clear idea of what you will change to prevent this inefficiency.

ProxyExt properties

Regardless of the driver, the proxy extension (ProxyExt) for any proxy point has a number of *core properties* that provide the same behavior. Depending on the driver, other ProxyExt properties are often present—see the particular driver document for details on those properties.

Core properties in any ProxyExt include the following:

- **Status**
  (read only) Status of the proxy extension, which in turn determines parent point status. See "Proxy point status" on page 1-33.
- **Fault Cause**
  (read only) If point has fault status, provides text description why.
- **Enabled**
  Either true (default) or false. While set to false, the point's status becomes disabled and polling is suspended.
- **Device Facets**
  (read only) Native facets used in proxy read or writes (Parent point's facets are used in point status display, and are available for edit in Add and Edit dialogs in Point Manager).
- **Conversion**
  Specifies the conversion used between the "read value" (in Device Facets) and the parent point's output (in selected point facets).
  Note: *In most cases, except when working with particular Ndio proxy points, or perhaps Modbus proxy points, the standard "Default" conversion selection is best.*

  Conversion selections include:
  - Default
    (The default selection). Conversion between "similar units" is automatically performed within the proxy point. For example, if a Lon proxy point with a LonFloatProxyExt (Device Facets of degrees C), if you set the point's Facets to degrees F, its output value automatically adjusts.
    If you set the parent point's Facets to *dissimilar units* (say, in this case to kilograms), the parent point has a *fault status* to indicate a configuration error.
  - Linear
    Applies to certain Ndio proxy points, such as NdioVoltageInput and NdioResistiveInput. Also commonly used in some Modbus proxy points. For these points, you typically want the point's

output value in some units other than Device Facets (if Ndio, voltage or resistance), and the Linear selection provides two fields to make the transition:
  - Scale: Determines linear response slope.
  - Offset: Offset used in output calculation.
- Reverse Polarity
  Useful in a Boolean-type Ndio proxy point to reverse the logic of the hardware binary input. Also, may be used in *any* driver's proxied BooleanPoint, as a way to "flip" the read state of the native value.
  Note: *Be careful in the use of the reverse polarity conversion, as it may lead to later confusion when troubleshooting logic or communications problems.*
- Thermistor Type 3
  Applies to an NdioThermistorInput point, where this selection provides a "built-in" input resistance-to-temperature value response curve for Type 3 Thermistor temperature sensors.
- Tabular Thermistor
  Applies to an NdioThermistorInput point, where this selection provides a control for a popup dialog for a custom resistance-to-temperature value response curve, including ability to import and export response curves.

- **Tuning Policy Name**
  (most drivers) Associated network tuning policy for this proxy point, by name. Tuning policies are used to separate poll rates and other driver features. See "About Tuning Policies" on page 1-8.
- **Read Value**
  (read only) Last value read from the device, expressed in device facets.
- **Write Value**
  (read only) Applies if writable point only. Last value written, using device facets.

Proxy point status

In addition to status flags that also apply to non-proxy points (refer to the *User Guide* section "How status flags are set"), some status flags in proxy points are set and cleared resulting from polled value or status updates received via the parent driver's communications subsystem, for example:

- **down**
  Driver unable to receive reply from parent device, as per configuration in Monitor extension. In this case, all proxy point children of the device will have status down.
- **fault**
  Typically, this indicates an AX configuration error or license error. If a fault occurs following normal (ok) status, it could be a "native fault" condition detected within the device, or perhaps some other fault criteria that was met. The point's proxy extension contains a "Fault Cause" text property that contains more information.
- **stale**
  Since the last poll update, the point's value has not updated within the specified "Stale Time" of its Tuning Policy. This stale status clears upon next received poll value.

Statuses above are set automatically, but you typically set a fourth status for a proxy point *manually*:

- **disabled**
  Using the proxy extension's Boolean property "Enabled" (default is true). While set to false (disabled), polling stops for that point, as well as any further status changes.

Note: *Typically, a proxy point has an* `alarm` *status only if its value falls in the "offnormal algorithm" limits specified in its own (NiagaraAX) alarm extension. If the driver and protocol has "intrinsic" or "native" alarm determination methods (such as with BACnet), a proxy point may show alarm status that originated locally within that device. In the case of the Niagara Bacnet driver, this also applies to* `override` *status, possibly a "native" condition for a proxied BACnet object.*

Effect of facets on proxy points

Each proxy point typically represents a piece of data in a remote device (see "About proxy points" on page 1-27). In cases for drivers that support a "learn" mechanism, a proxy point may be created with its facets (units) already defined—these reflect its "Device Facets" (in its ProxyExt).

If needed, you can change the *parent* proxy point's facets to use *similar* (same family) units. For example, if a Lonworks proxy point for a temperature NVI (learned in degrees C), you can change the proxy points facets to degrees F. The "default" conversion performed in the LonProxyExt automatically adjusts the output value appropriately. In this case, facets directly affect the point's out value.

Note: *If you change a proxy point's facets to use units that are dissimilar from the device facets in its ProxyExt (without also changing its Conversion from "Default"), the proxy point will have a fault status (to show misconfiguration). For example, if its device facets are temperature, and you change the point's facets to pressure, the point will have a fault status.*

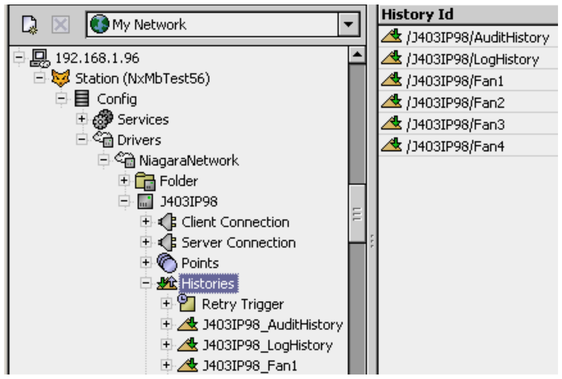
For details on properties common to the ProxyExt in all proxy points, including Device Facets and Conversion, see "ProxyExt properties" on page 1-32.

About the Histories extension

Not all drivers have devices with a HistoryDeviceExt—notably, this exists only if the driver protocol (or device) provides *local data logging*. Currently, a **`Histories`** device extension exists under components NiagaraStation, ObixClient, R2ObixClient, and BacnetDevice (under a BacnetDevice, this extension is named "Trend Logs").

Note: *"Database device" components in any of the RdbmsNetwork drivers (rdbSqlServer, rdbMySQL, etc.) also each have a* **`Histories`** *extension. See "Importing and exporting data using the RdbmsNetwork" in the* Rdbms Driver Guide *for related details.*

Figure 1-45 *Histories device extension (NiagaraStation)*



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, see "About the Retry Trigger" on page 1-34.

The difference between a history import and history export are as follows:

- **Import**
  Any imported history appears in the local station as a Niagara history, where data is effectively "pulled" from a remote *source* device.
  - For example, if a history import descriptor under Histories of a NiagaraStation device, the source is another Niagara history in that remote station.
  - If a history import descriptor under Trend Logs of a BacnetDevice, the source is a BACnet Trend Log object residing in that BACnet device.
- **Export**
  Applies to a NiagaraStation (or "database device" under an RdbmsNetwork). An exported history is a Niagara history that 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).
  Note: *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. See the* BACnet Guide *section "Bacnet server configuration overview" for more details.*

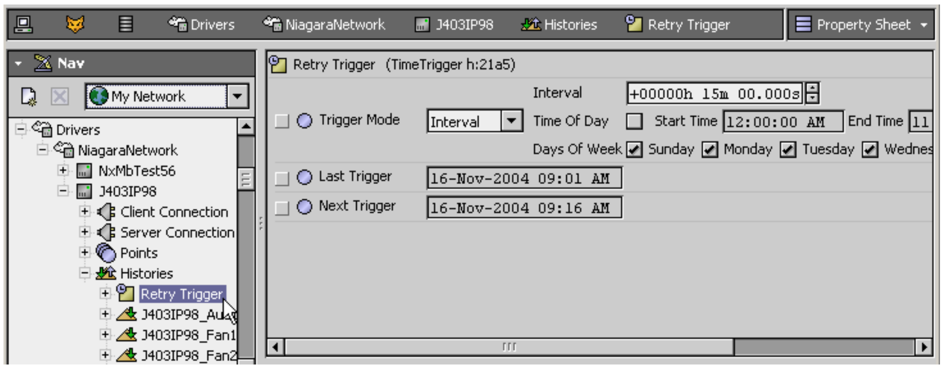
You create history import descriptors and export descriptors under the Histories extension of a device using *separate views*. For details, see "About Histories extension views" on page 1-46.

About the Retry Trigger

By default, device extensions like **`Histories`**, **`Schedules`**, and others contain a "Retry Trigger"-named component, appearing in the Nav tree but not in any Manager view (Figure 1-46).

Note: *Retry Trigger is unique in that it requires no linking of its output for operation.*

Figure 1-46 Retry Trigger under device Histories extension



Retry Trigger (TimeTrigger h:21a5)

Trigger Mode: Interval — Interval +00000h 15m 00.000s; Time Of Day: Start Time 12:00:00 AM, End Time 11; Days Of Week: Sunday, Monday, Tuesday, Wednes
Last Trigger: 16-Nov-2004 09:01 AM
Next Trigger: 16-Nov-2004 09:16 AM

Upon any *unsuccessful execution* of a descriptor component (i.e. history import descriptor, schedule import descriptor, and so on) contained in that same device extension, the **`Retry Trigger`** defines subsequent “retries” that will automatically occur upon the interval period defined (default value is 15 minutes). This continues until successful execution occurs.

About the Alarms extension

Not all drivers have devices with an AlarmDeviceExt (Alarms)—notably, this exists only if the driver protocol (or device) provides for *local (native) alarming*. Currently, only a device component under a NiagaraNetwork (station), BacnetNetwork, or ObixNetwork has a child **`Alarms`** extension.

Alarms extension properties in devices NiagaraStation and BacnetDevice (drivers NiagaraDriver and Bacnet, respectively) specify how alarms from that device are mapped into the station’s own alarm subsystem, plus provide status properties related to alarm sharing. Unlike some other device extensions, there are no special views, nor does the extension 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 AlarmClass used 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 typically discover, add, and manage alarm feeds. See the *NiagaraAX oBIX Guide* for more details.

Alarms extension properties

The following properties apply to the Alarms device extension under a NiagaraStation or BacnetDevice:

- **Alarm Class**
  Under a BacnetDevice, or using the *first* Alarm Class property option (**`Replace`**) under a NiagaraStation, provides a selection list of a local AlarmClasses, from which you can select *one* to use for all alarms received from this device.
  Included is a “**`Use Existing`**” checkbox option, where:
  - Alarms from this remote station are routed to any “matching” AlarmClass, that is, one with *identical* name as the “alarmClass” field in each alarm record. If no local “matching” AlarmClass is found, the station's *default* AlarmClass is used.
  - If the checkbox is cleared, all received alarms are routed to the single local AlarmClass specified.

  Two additional Alarm Class property options are available for the Alarm Class property of the NiagaraStation Alarms extension:
  - **`Prepend`** — To add *leading text* (as specified) to the incoming “alarmClass” field string, then route to any local “matching” Alarm Class in the station.
  - **`Append`** — To add *trailing text* (as specified) to the incoming “alarmClass” field string, then route to any local “matching” Alarm Class in the station.

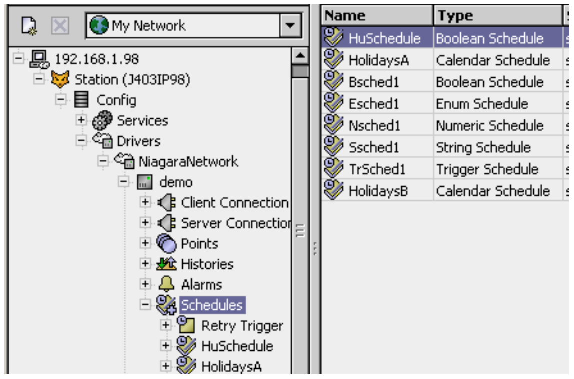
  For more details, see “Prepend and Append alarm routing notes” on page 2-27.
- **Last Received Time**
  Timestamp when last alarm from this device was *received*. This is configured remotely, either in the sending Niagara station or the BACnet device.
  Note: *Remaining properties apply to the Alarms extension under a NiagaraStation component only (are not available in Alarms extension under a BacnetDevice).*

- **Source Name**
  Available in the Alarms extension of a NiagaraStation, so that the name of the sending station can be added to the received alarm's "alarm source" name, or another format be selected. This affects the alarm's *display* (versus routing).
  The available choices are:
  - **`Prepend`** — (default) To add *leading text* (as specified) to the incoming "alarmSource" field string, where the default format is `%parent.parent.displayName%:`
    For example, an alarm in the remote station named "EastWing" for an alarm record with its local alarm source of "Room101", the resulting alarm source would look like `EastWing:Room101`
  - **`Append`** — To add *trailing text* (as specified) to the incoming "alarmSource" field string. For example, if the format entered is `-%parent.parent.displayName%`
    And the alarm in the remote station named "EastWing" is for an alarm record with its (local) alarm source of "Room101", the resulting alarm source would be `Room101-EastWing`
  - **`Use Existing`** — Only uses the incoming "alarmSource" field string (ignores whatever format text is entered in this property).
  - Replace — Only uses the format text entered in this property as alarm source (ignoring whatever incoming "alarmSource" string is received).
- **Last Send Time**
  Timestamp when 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 Failure Time**
  Timestamp when last local alarm routed to this station could not be sent.
- **Last Send Failure Cause**
  Text string describing failure cause routing local alarm to this station.

About the Schedules extension

Not all drivers have devices with a ScheduleDeviceExt (Schedules)—notably, this exists only if the driver protocol (or device) provides *local event scheduling*. Currently, this device extension exists only under components NiagaraStation and BacnetDevice.

Figure 1-47 *Schedules device extension (NiagaraStation)*



My Network

- 192.168.1.98
  - Station (J403IP98)
    - Config
      - Services
      - Drivers
        - NiagaraNetwork
          - demo
            - Client Connection
            - Server Connection
            - Points
            - Histories
            - Alarms
            - Schedules
              - Retry Trigger
              - HuSchedule
              - HolidaysA

Name	Type
HuSchedule	Boolean Schedule
HolidaysA	Calendar Schedule
Bsched1	Boolean Schedule
Esched1	Enum Schedule
Nsched1	Numeric Schedule
Ssched1	String Schedule
TrSched1	Trigger Schedule
HolidaysB	Calendar Schedule

A device's Schedules extension is the parent container for *imported schedules*—Niagara schedule components with a ScheduleImportExt. Events in an imported schedule are obtained from that device, and are *read-only* (often called "slave schedules"). By default, the Schedules extension also contains a Retry Trigger, see "About the Retry Trigger" on page 1-34.

The Schedules extension can also contain *schedule export descriptors*. These correspond to *local station* schedules that are exported ("pushed") to that remote station (often called "master schedules"). A schedule export descriptor is automatically created whenever a local schedule is "imported" into a *remote station*.

The difference between a schedule import and schedule export are as follows:

- **Import**
  An imported schedule appears in the local station as a Niagara schedule, where read-only schedule events are configured/adjusted in a remote source device.
- If under a NiagaraStation device, the source is a Niagara schedule in that remote station.
- If under a BacnetDevice, the source is a BACnet Schedule or Calendar object residing in that BACnet

device. That object's data is now modeled as a Niagara schedule component.

- **Export**
  This is a Niagara schedule in the local station that is *exported into* a remote station (NiagaraNetwork) or BACnet Schedule or Calendar object (BacnetNetwork). A resulting schedule export descriptor allows configuration of a "push" mechanism to keep event configuration synchronized in the remote device.
  Note: *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 "exposing" Niagara schedule components in the station as either a BACnet Schedule object or Calendar object. For details, see the Bacnet Users Guide.*

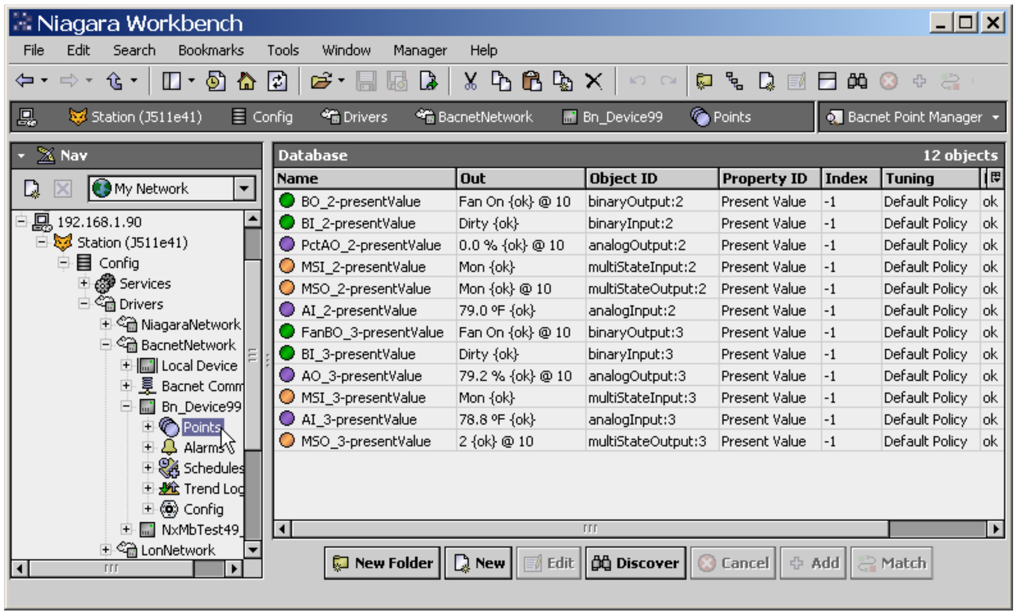
You create imported schedules under a device's Schedules extension using an Import Manager view. A separate Export Manager view provides access to schedule export descriptors. For details, see "About Schedules extension views" on page 1-49.

Note: *Schedule components local to the station can reside anywhere under the station's Config hierarchy and be imported by one or more other stations. As this occurs, a schedule export descriptor is automatically created under the NiagaraStation component that represents the remote station (and is located in its Schedules container). On the other hand, if you import a schedule from another NiagaraStation or BacnetDevice, it must reside in that device's Schedule container. Imported schedules are always under a specific device.*

About the Point Manager

The Point Manager is the *default view* for the *Points* extension under any device object. Like other manager views, it is table-based (Figure 1-48). Here, each row represents a *proxy point* (or a points folder) under Points.

Figure 1-48 *Point Manager under a BacnetDevice*



Database 12 objects

Name	Out	Object ID	Property ID	Index	Tuning	
● BO_2-presentValue	Fan On {ok} @ 10	binaryOutput:2	Present Value	-1	Default Policy	ok
● BI_2-presentValue	Dirty {ok}	binaryInput:2	Present Value	-1	Default Policy	ok
● PctAO_2-presentValue	0.0 % {ok} @ 10	analogOutput:2	Present Value	-1	Default Policy	ok
● MSI_2-presentValue	Mon {ok}	multiStateInput:2	Present Value	-1	Default Policy	ok
● MSO_2-presentValue	Mon {ok} @ 10	multiStateOutput:2	Present Value	-1	Default Policy	ok
● AI_2-presentValue	79.0 ºF {ok}	analogInput:2	Present Value	-1	Default Policy	ok
● FanBO_3-presentValue	Fan On {ok} @ 10	binaryOutput:3	Present Value	-1	Default Policy	ok
● BI_3-presentValue	Dirty {ok}	binaryInput:3	Present Value	-1	Default Policy	ok
● AO_3-presentValue	79.2 % {ok} @ 10	analogOutput:3	Present Value	-1	Default Policy	ok
● MSI_3-presentValue	Mon {ok}	multiStateInput:3	Present Value	-1	Default Policy	ok
● AI_3-presentValue	78.8 ºF {ok}	analogInput:3	Present Value	-1	Default Policy	ok
● MSO_3-presentValue	2 {ok} @ 10	multiStateOutput:3	Present Value	-1	Default Policy	ok

When building a network in the station, you use this view to create, edit, and delete proxy points in the station database. See "About proxy points" on page 1-27.

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.*

At the bottom of the view, buttons "New Folder" and "New" let you create new point folders and proxy points in the station database. An "Edit" button lets you edit one or more selected proxy points. If the driver supports online "device learning," buttons "Discover," and "Match" are also typically available. Finally, additional buttons may be available, depending on the driver.

For more details, see:

- Points New Folder and New
- Point Edit
- About Point Discover, Add and Match (Learn Process)
- About other Points views

Note: *Also see "Niagara Point Manager notes" on page 2-20 for details specific to the Point Manager view of a Points device extension under a NiagaraStation device component (NiagaraNetwork).*

Points New Folder and New

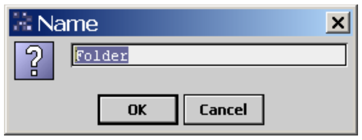
Buttons New Folder and New exist in a device's Point Manager view under *any* driver network.

Note: *New Folder and New are tools on the Point Manager toolbar, and in the Manager menu.*

New Folder

New Folder in the Point Manager adds a special "PointFolder" component that you can use to organize proxy points. This is equivalent to copying that same component from that driver's palette. A Name dialog lets you name the folder before it is created (Figure 1-49).

Figure 1-49 *Example New Folder dialog in Point Manager*



When you click OK, a new PointFolder component is added under Points. Point folders are often useful, especially if you are creating many proxy points, or need extra wire space for additional kitControl or schedule components (and whatever links you wish to create between them).

Note: *A devices's PointFolder component is different than a "normal" Folder component, because it includes that devices's Point Manager view as its default view (just like the parent Points component). You can double-click a point folder (either in the Point Manager view pane or in the Nav tree), to access its Point Manager. Also, an "All Descendants" command is available from the root Points Manager, which allows you to see all proxy points in that device.*

All Descendants When in the Point Manager of a point folder, it is "unaware" of proxy points in *other* point folders (or in the root of Points). However, from the *root* Point Manager view (of Points), you can "flatten" the folder structure to view *all proxy points* by selecting "All Descendants" from the Manager menu (Figure 1-50) or simply clicking the "All Descendants" tool on the toolbar (Figure 1-51). Note that "all descendants" is also available at *any* point folder (level) as well—you just see all points in descendants from that folder level on down.

Note: *Be aware that in a device with many point folders and proxy points, using the all descendants feature (particularly at the Points root level) may decrease performance. This might result from so many points becoming subscribed.*

Figure 1-50 *Manager menu "All Descendants" command*

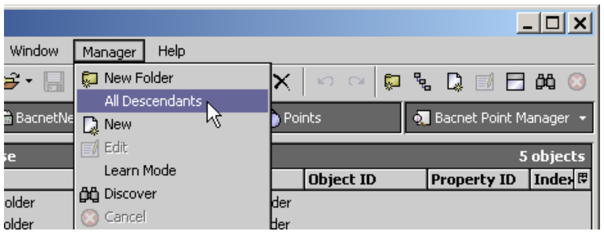
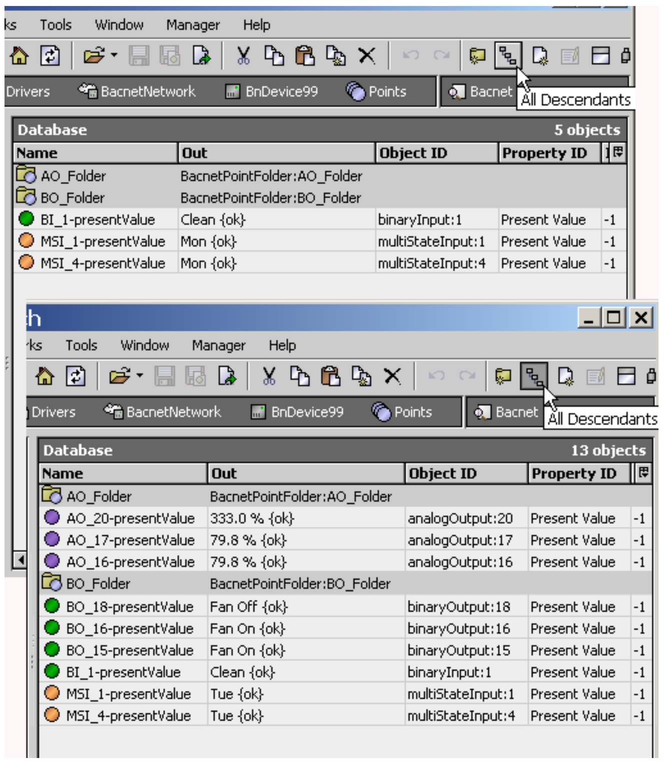


Figure 1-51 *Point Manager "All Descendants" tool on toolbar.*



Database — 5 objects

Name	Out	Object ID	Property ID	
AO_Folder	BacnetPointFolder:AO_Folder			
BO_Folder	BacnetPointFolder:BO_Folder			
BI_1-presentValue	Clean {ok}	binaryInput:1	Present Value	-1
MSI_1-presentValue	Mon {ok}	multiStateInput:1	Present Value	-1
MSI_4-presentValue	Mon {ok}	multiStateInput:4	Present Value	-1

Database — 13 objects

Name	Out	Object ID	Property ID	
AO_Folder	BacnetPointFolder:AO_Folder			
AO_20-presentValue	333.0 % {ok}	analogOutput:20	Present Value	-1
AO_17-presentValue	79.8 % {ok}	analogOutput:17	Present Value	-1
AO_16-presentValue	79.8 % {ok}	analogOutput:16	Present Value	-1
BO_Folder	BacnetPointFolder:BO_Folder			
BO_18-presentValue	Fan Off {ok}	binaryOutput:18	Present Value	-1
BO_16-presentValue	Fan On {ok}	binaryOutput:16	Present Value	-1
BO_15-presentValue	Fan On {ok}	binaryOutput:15	Present Value	-1
BI_1-presentValue	Clean {ok}	binaryInput:1	Present Value	-1
MSI_1-presentValue	Tue {ok}	multiStateInput:1	Present Value	-1
MSI_4-presentValue	Tue {ok}	multiStateInput:4	Present Value	-1

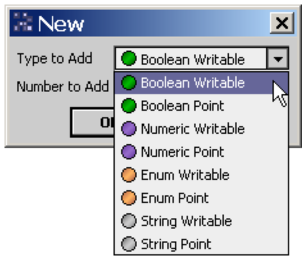
Note: *If you are using point folders, and you click on a table to column to resort points, please be aware that any proxy point-to-point folder organization is lost during that view. However, you can always see contents of point folders clearly in the Nav tree, and again when returning to the Point Manager view from another view.*

New

New in the Point Manager is to add new proxy points to the station. Typically, you use New only if the driver does not support online discovery, *or* if you are programming offline. Under any of the Modbus drivers, for example, you use New to add proxy points.

Using New is equivalent to copying proxy points from that driver's palette, except it provides *more utility* to specify other parameters. Minimally, the New dialog provides a selection list for proxy point *type*, and also the number of points to add (Figure 1-52).

Figure 1-52 *Example New dialog in Point Manager*



New

Type to Add: Boolean Writable

Number to Add

In some driver networks, the New dialog may provide other parameters, such as a starting address range when adding multiple proxy points. When you click OK, the number and type of proxy points you specified are added under Points or a point folder. You need to edit specific properties for each new proxy point (note these are typically properties of its *proxy extension*).

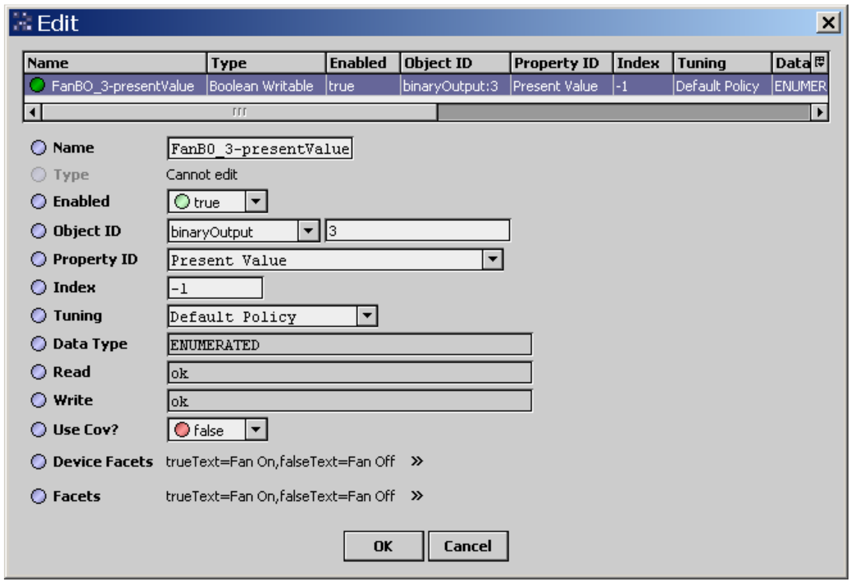
Point Edit

In the Point Manager view, you can Edit any proxy point shown in the station database by simply double-clicking it. (Edit does not apply to point folders.)

Edit

The Edit dialog appears with the proxy point listed (Figure 1-53).

Figure 1-53 *Example Edit dialog in Point Manager (single point)*


Edit

Name	Type	Enabled	Object ID	Property ID	Index	Tuning	Data
FanBO_3-presentValue	Boolean Writable	true	binaryOutput:3	Present Value	-1	Default Policy	ENUMER

Name	FanBO_3-presentValue
Type	Cannot edit
Enabled	true
Object ID	binaryOutput 3
Property ID	Present Value
Index	-1
Tuning	Default Policy
Data Type	ENUMERATED
Read	ok
Write	ok
Use Cov?	false
Device Facets	trueText=Fan On,falseText=Fan Off
Facets	trueText=Fan On,falseText=Fan Off

Note: *The Edit dialog for a proxy point shows mostly properties under its proxy extension, plus (typically) the parent point's Name and Facets. Many of the proxy extension values are required for communications, and will vary among drivers. To access all properties of the proxy point, including all those under any of its extensions, go to its property sheet.*

When a single point is selected in the Edit dialog, you can edit any property except Type *(fixed when you added the point). Included is the ability to edit the Name of the proxy point in the station. This is equivalent to the right-click Rename command on the point.*

The following related topics also apply:

- Proxy point "gang" edits
- Manager table features

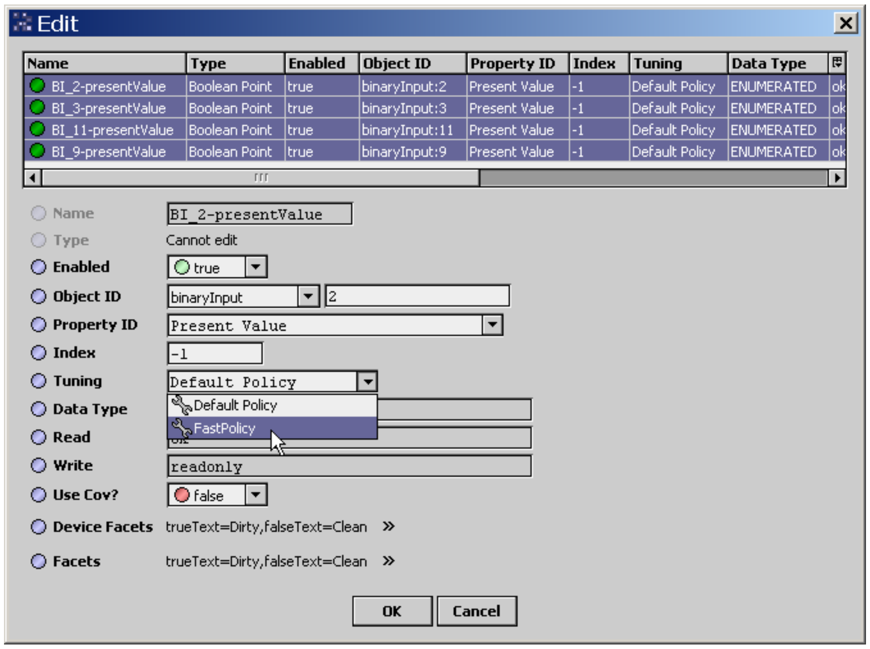
Proxy point "gang" edits

The Edit button in the Point Manager allows you to edit one or more proxy points in the station in a single dialog. Before clicking Edit, use standard Windows keyboard controls to highlight (select) *multiple points* (e.g. hold down Ctrl key and click desired points).

Note: *Edit is also on the Point Manager toolbar and the Manager menu, if any point is selected.*

The "gang" edit feature is useful for making the same change in multiple (selected) proxy points. For example, as shown in Figure 1-54, you can change "Tuning" (point's associated tuning policy) in multiple points at the same time (versus editing this individually).

Figure 1-54 *Example "gang edit" in Edit dialog in Point Manager*



Name	Type	Enabled	Object ID	Property ID	Index	Tuning	Data Type	
BI_2-presentValue	Boolean Point	true	binaryInput:2	Present Value	-1	Default Policy	ENUMERATED	ok
BI_3-presentValue	Boolean Point	true	binaryInput:3	Present Value	-1	Default Policy	ENUMERATED	ok
BI_11-presentValue	Boolean Point	true	binaryInput:11	Present Value	-1	Default Policy	ENUMERATED	ok
BI_9-presentValue	Boolean Point	true	binaryInput:9	Present Value	-1	Default Policy	ENUMERATED	ok

Name	BI_2-presentValue
Type	Cannot edit
Enabled	true
Object ID	binaryInput 2
Property ID	Present Value
Index	-1
Tuning	Default Policy
Data Type	
Read	
Write	readonly
Use Cov?	false
Device Facets	trueText=Dirty,falseText=Clean
Facets	trueText=Dirty,falseText=Clean

When you have the Edit dialog open with multiple points, you can also click on a specific one to make individual edits (e.g. Name, or any other editable property), during the same dialog session where you made "gang" property edits. When you click OK, all the changes are applied to the proxy points as made during your Edit session. For more details, see "About gang edits."

Note: *When you have multiple points selected in the Edit dialog, properties that must be unique (such as Name) are automatically unavailable (dimmed). However, note that some properties that typically should be unique (often address properties) may still be available for "gang edit," as this rule is not automatically enforced. Typically, when editing these properties, you should verify only a single point component (row) is highlighted in the table.*

About Point Discover, Add and Match (Learn Process)

Online "point learns" are possible in some driver networks, for example the NiagaraNetwork, Bacnet-Network, LonNetwork, and NdioNetwork. Whenever available, this method is the easiest way to accurately add proxy points to the station database.

Any proxy point learn in NiagaraAX is a two-step *process* using the Point Manager, where you:

1. Under a selected device component, use its (Points device extension) Point Manager view to Discover data items as point *candidates* for inclusion in the station database.
2. Select and Add from those candidates, creating proxy points under the device's Points container. (If you already used New to add points, you can also Match to those existing points).

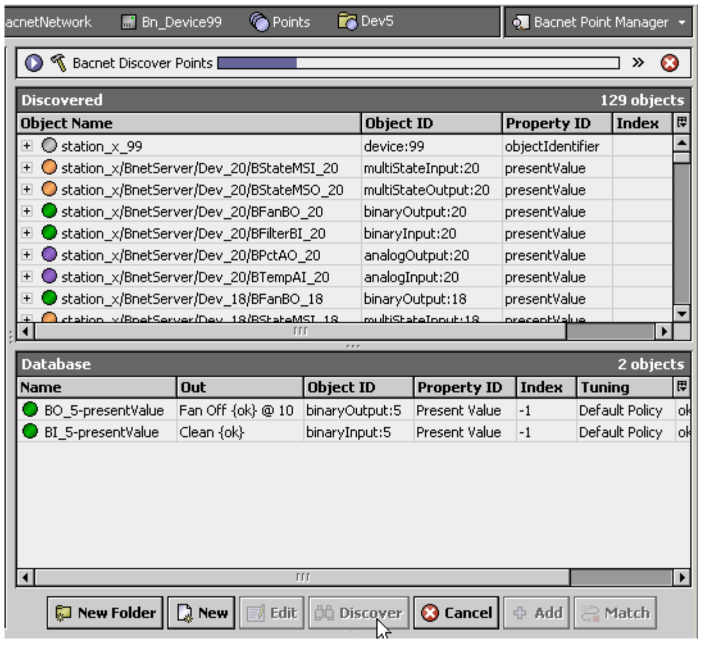
Note: *The Point Manager reinforces this process by providing two separate panes in the view whenever you enter "Learn Mode." See "About Learn toggle" on page 1-18.*

Discover

The Discover button is available in the Point Manager only if the driver supports online discovery. When you click Discover, the Point Manager view splits into two panes (*Learn Mode*), and at the same time typically launches a discovery "job" (Figure 1-55).

Note: *Under a LonNetwork, Point Manager has a Learn Mode, but no point Discover. All possible data items were already discovered when the parent device was discovered and added. Here, enter Learn Mode by toggling (see "About Learn toggle" on page 1-18).*

Figure 1-55 *Discover splits Point Manager view*



Discovered 129 objects

Object Name	Object ID	Property ID	Index
station_x_99	device:99	objectIdentifier	
station_x/BnetServer/Dev_20/BStateMSI_20	multiStateInput:20	presentValue	
station_x/BnetServer/Dev_20/BStateMSO_20	multiStateOutput:20	presentValue	
station_x/BnetServer/Dev_20/BFanBO_20	binaryOutput:20	presentValue	
station_x/BnetServer/Dev_20/BFilterBI_20	binaryInput:20	presentValue	
station_x/BnetServer/Dev_20/BPctAO_20	analogOutput:20	presentValue	
station_x/BnetServer/Dev_20/BTempAI_20	analogInput:20	presentValue	
station_x/BnetServer/Dev_18/BFanBO_18	binaryOutput:18	presentValue	
station_x/BnetServer/Dev_18/BStateMSI_18	multiStateInput:18	presentValue	

Database 2 objects

Name	Out	Object ID	Property ID	Index	Tuning	
BO_5-presentValue	Fan Off {ok} @ 10	binaryOutput:5	Present Value	-1	Default Policy	ok
BI_5-presentValue	Clean {ok}	binaryInput:5	Present Value	-1	Default Policy	ok

Note: *Under a NiagaraNetwork (only), a Niagara Point Manager discover produces an intermediate dialog: the Bql Query Builder. You use it to browse the remote station and specify what items to select from as discovered proxy point candidates. For details, see "About the Bql Query Builder" on page 2-21.*

In Learn Mode, the two panes in the Point Manager operate as follows:

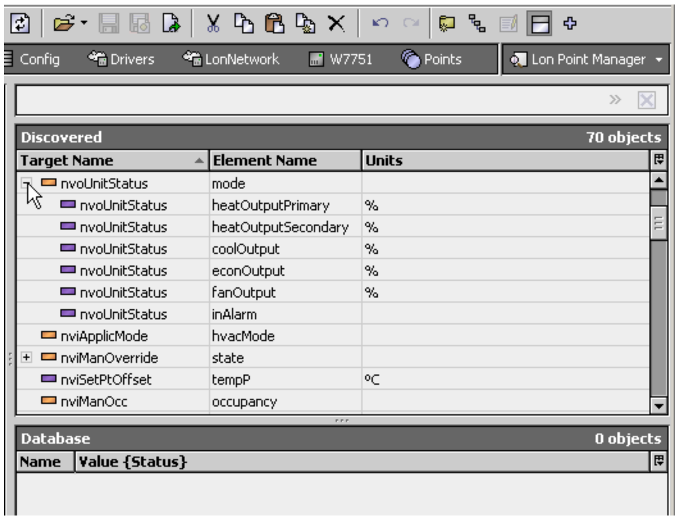
- **Discovered (top pane)**
  Lists data items discovered on the driver's network, as proxy point candidates. For any data item found *that already exists* in the station database, it will appear "ghosted" (listed faintly). Note items listed may be "expandable"—see "Discovered selection notes" on page 1-42.
  A job "progress bar" is also included on top of the Discovered pane.
  Note: *A Cancel button is available during a discovery job. If needed, use it to stop discovery.*
- **Database (bottom pane)**
  Lists proxy points and point folders that are currently in the station database.

Note: *As necessary, drag the border between the two panes to resize. Also (at any time), toggle between the two-pane Learn Mode and the single-pane (Database) view by clicking the Learn Mode tool in the toolbar (Figure 1-24 on page 18), or using the Learn Mode command in the Manager menu.*

Discovered selection notes Often, data items listed in the Point Manager's discovered pane are expandable, having one or more *related items*, each individually selectable. Expandable items are indicated by a leading plus (+), which you click to expand (a toggle control).

Figure 1-56 shows an example item (Lonworks nvoUnitStatus) expanded to reveal individual numeric-type elements, each selectable as a separate proxy point.

Figure 1-56 *Expand discovered data items to see all point candidates*



Discovered — 70 objects

Target Name	Element Name	Units
nvoUnitStatus	mode	
nvoUnitStatus	heatOutputPrimary	%
nvoUnitStatus	heatOutputSecondary	%
nvoUnitStatus	coolOutput	%
nvoUnitStatus	econOutput	%
nvoUnitStatus	fanOutput	%
nvoUnitStatus	inAlarm	
nviApplicMode	hvacMode	
nviManOverride	state	
nviSetPtOffset	tempP	°C
nviManOcc	occupancy	

Database — 0 objects

Name	Value {Status}

Here, if you selected only the top “mode” element to add, you would have one proxy EnumPoint to monitor the enumerated unit status (auto, heat, cool, etc), but would *not have* any of the related numeric-type items proxied as control points.

Depending on the driver/device type, expandable discovered items represent individual properties or other “structured” data. Some examples:

- BacnetDevice—Each top item is typically the “present value” property of the BACnet object (most commonly selected). Expand the item to see other properties of the object.
- NiagaraStation—Using Bql Query Filter defaults (`Config`, `control`, `ControlPoint`), each top item is equivalent to the “Out” slot of the Niagara component (and most commonly selected). Expand the item to see other slots in the component (including Out).
- LonDevice—Each top item is typically the *first* element (field) in a structured SNVT (multi-field data structure), as used in that particular network variable (nv or nci). To access other data fields in the SNVT’s structure, you must expand that item.

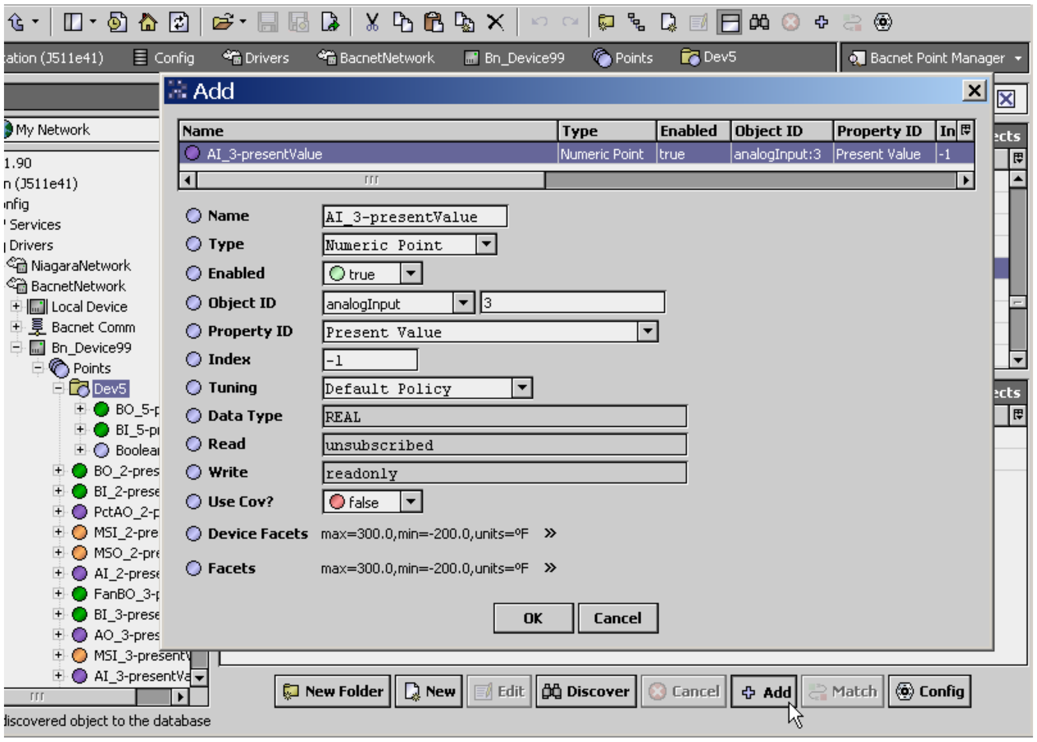
For specific details, refer to the “Point discovery notes” section in a particular driver document.

Add

The Point Manager's Add button is available in Learn Mode when you have one or more data items selected (highlighted) in the top discovered pane. When you click Add, an Add dialog appears that allows you to edit properties before the proxy point(s) are created in the station (Figure 1-57).

Note: *Whenever you select one or more items in the top discovered pane, the toolbar also has an available Add tool (“plus” symbol), and the Manager menu has an Add command. Also, you can simply double-click a discovered item to bring it up in the Add dialog.*

Figure 1-57 *Add dialog from Add button in Point Manager*



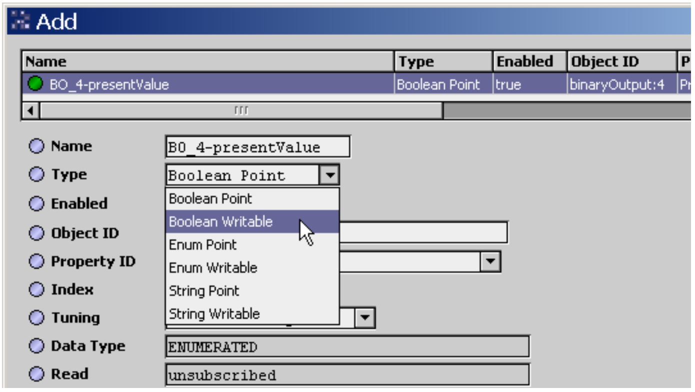
Name	Type	Enabled	Object ID	Property ID	In
AI_3-presentValue	Numeric Point	true	analogInput:3	Present Value	-1

Name	AI_3-presentValue
Type	Numeric Point
Enabled	true
Object ID	analogInput 3
Property ID	Present Value
Index	-1
Tuning	Default Policy
Data Type	REAL
Read	unsubscribed
Write	readonly
Use Cov?	false
Device Facets	max=300.0,min=-200.0,units=°F
Facets	max=300.0,min=-200.0,units=°F

The Add dialog is nearly identical to the point Edit dialog, but allows you to edit Type as well as other properties.

Often, you may wish to *change* Type from the pre-selected one, at least between read-only points and the equivalent writable control point within that data category. For example, if adding a proxy point for the present value (default) property for a BACnet Binary Output object, you may wish it to be a read-only BooleanPoint point rather than the default BooleanWritable. As shown in Figure 1-58, you can do this in the Add dialog *before* it is added to the station database, (but not later using the point Edit feature).

Figure 1-58 *Change Type as needed in point Add dialog*



Name	Type	Enabled	Object ID	P
BO_4-presentValue	Boolean Point	true	binaryOutput:4	Pr

Name	BO_4-presentValue
Type	Boolean Point
Enabled	
Object ID	
Property ID	
Index	
Tuning	
Data Type	ENUMERATED
Read	unsubscribed

Note: *In most cases, alternate point Types include StringPoint, and possibly others. Generally speaking, there are few practical applications in changing the data category of a proxy point type (e.g. from Boolean to Enum or Sting), however, this may be an option. Note that if working under a NiagaraNetwork, only read-only proxy points are available.*

Address-related properties in the Add point dialog already have acceptable values for operation (otherwise, the data item would not have been discovered). It is possible you change only Name and possibly Type, unless you know other settings you wish to change now. You can always Edit these properties in the proxy point(s) after you click OK and add them to your station.

Match

Match is a feature that may be useful when you have an application with proxy points you wish to reuse, or if you have programmed offline using the New point feature.

- In the first case (application for reuse), you could have some number of proxy points included in an application that you have saved and now recopied under the target Points container. Often, address-related properties in the copied proxy points are incorrect. However, you can use the Point Manager's Learn Mode and step through each proxy point in the copied application, and use the Match feature to "sync" with the intended (and discovered) data item.
- In the second case (offline programming) where a connection to the actual device network is unavailable, you can manually add New devices and New proxy points, and begin station engineering of a driver network. Typically, most component creations under a driver network are possible (including all levels) using the New command in the various "manager" views (Device Manager, Point Manager, other device extension managers). Or, you can add saved applications (from the device level on down) and edit as necessary. Then, when online with the driver network later, you could use Match to "sync" to existing components (device-level, proxy points, and so forth).

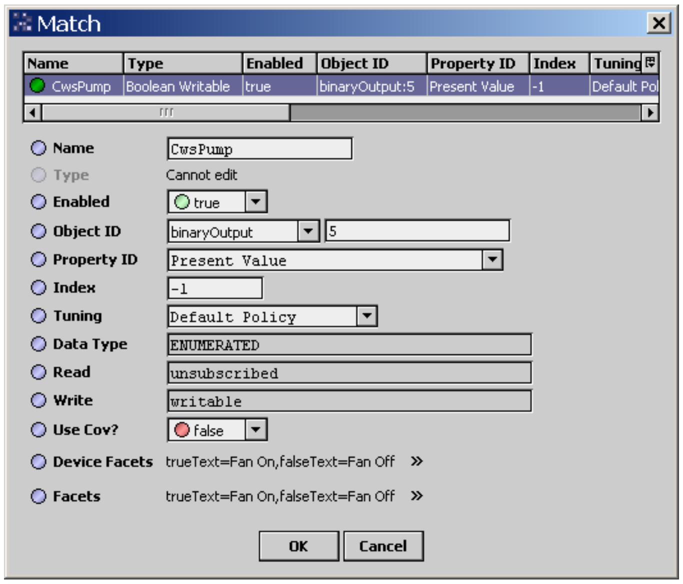
The Match button in the Point Manager becomes available when in Learn Mode, and you have:

1. Selected *one* point candidate in the *top* (Discovered) pane.
2. Selected *one* existing proxy point in the *bottom* (Database) pane.

Note: *In this case, the toolbar also has an available Match tool, and the Manager menu has a Match command.*

When you click Match, the Match dialog appears, as shown in Figure 1-59.

Figure 1-59 *Match dialog example in Point Manager*


Match

Name	Type	Enabled	Object ID	Property ID	Index	Tuning
CwsPump	Boolean Writable	true	binaryOutput:5	Present Value	-1	Default Pol

Name	CwsPump
Type	Cannot edit
Enabled	true
Object ID	binaryOutput 5
Property ID	Present Value
Index	-1
Tuning	Default Policy
Data Type	ENUMERATED
Read	unsubscribed
Write	writable
Use Cov?	false
Device Facets	trueText=Fan On,falseText=Fan Off
Facets	trueText=Fan On,falseText=Fan Off

Note: *Match is strictly a "one-to-one" function for "discovered-to-database"—note that it is unavailable any time you have multiple items selected either in either the top Discovered pane or bottom Database pane.*

The Match point dialog is nearly identical to the single-point Edit dialog, and typically provides the "discovered" point address values. Often, most properties in the Match dialog have acceptable *address* values required for operation (otherwise, the item would not have been discovered). You can always Edit the proxy point after you click OK and add it to your station.

About other 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).

About Histories extension views

For an overview of a device **`Histories`** extension, see "About the Histories extension" on page 1-34. Standard views available on a device's Histories extension include:

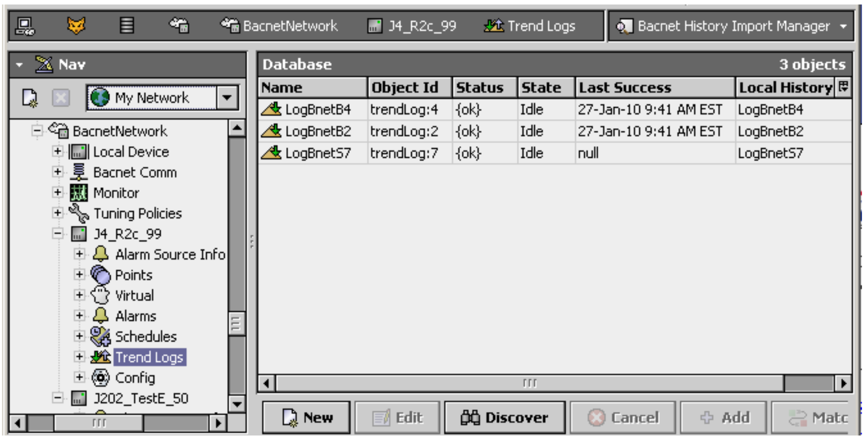
- History Import Manager
- Device Histories View (AX-3.5 and later)

Additional views may be available, depending on driver. The Histories extension of a NiagaraStation has a **`History Export Manager`** view. See "Niagara histories notes" on page 2-29.

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 (Figure 1-60). Here, each row is a history *import descriptor*. Each descriptor specifies how log data is imported (pulled) from the device into the station as a history.

Figure 1-60 *History Import Manager under a BacnetDevice (Trend Logs)*



Name	Object Id	Status	State	Last Success	Local History
LogBnetB4	trendLog:4	{ok}	Idle	27-Jan-10 9:41 AM EST	LogBnetB4
LogBnetB2	trendLog:2	{ok}	Idle	27-Jan-10 9:41 AM EST	LogBnetB2
LogBnetS7	trendLog:7	{ok}	Idle	null	LogBnetS7

You use this view to create, edit, and delete history import descriptors. Each import descriptor you add results in the creation of a local Niagara history—regardless if the source log data is a BACnet Trend Log, Niagara 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—any other components that may also reside under Histories do not appear. For example, you do not see the default "Retry Trigger" component (see "About the Retry Trigger" on page 1-34). However, you can use the Histories property sheet to access these items.*

At the bottom of the view, the "**`New`**" button lets you manually create new import descriptors in the station. An "**`Edit`**" button lets you edit one or more import descriptors. Buttons "**`Discover`**," "**`Add`**" and "**`Match`**" are also available, (these work similarly as in the Point Manager). An "**`Archive`**" button is available to manually import (pull data) into one or more selected histories.

Starting in AX-3.5, the **`History Import Manager`** in some drivers provides a "**`New Folder`**" button, to add folders to organize history import (and export) descriptors. Each such folder provides its own set of history manager views. See the section "Niagara History Import Manager" on page 2-30 for details on this, as well as other history import information specific to the niagaraDriver.

For more common details about a History Import Manager, see:

- History Import New
- History Import Edit
- About History Import Discover, Add and Match (Learn Process)

History Import New

Button **`New`** exists in a device's History Import view, but is typically used only if:

- Using "system tags" to import remote histories, such as in the **`Niagara History Import Manager`**, versus online discovery.
- Engineering offline—most devices with a **`Histories`** extension support online discovery (a File-Device is one exception). If offline, Match may be used later (when online with the device).

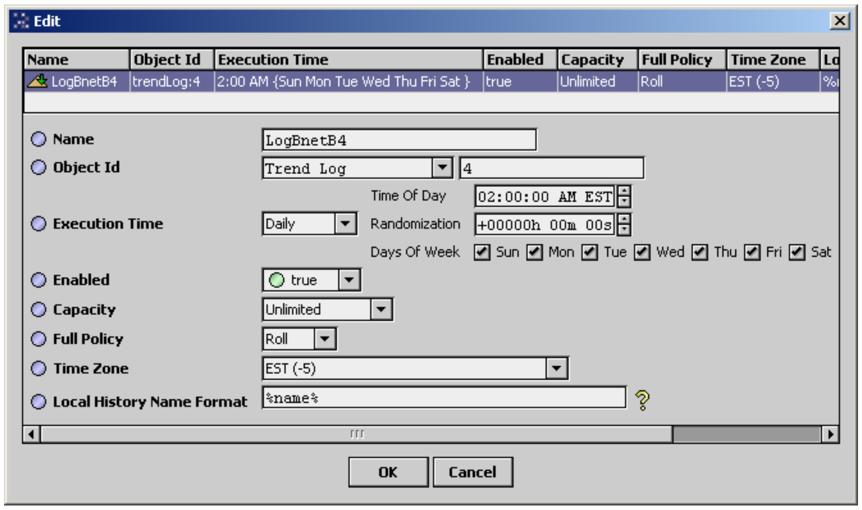
Note: *A "New" tool is also on the History Import Manager toolbar, and in the Manager menu.*

History Import Edit

In the History Import Manager, you can Edit any import descriptor in the station database by simply double-clicking it.

Edit The Edit dialog appears with the import descriptor listed (Figure 1-61).

Figure 1-61 *Edit dialog in Bacnet History Import Manager (single history)*



Edit

Name	Object Id	Execution Time	Enabled	Capacity	Full Policy	Time Zone	Lo
LogBnetB4	trendLog:4	2:00 AM {Sun Mon Tue Wed Thu Fri Sat }	true	Unlimited	Roll	EST (-5)	%|

Name	LogBnetB4	
Object Id	Trend Log	4
	Time Of Day	02:00:00 AM EST
Execution Time	Daily	Randomization +00000h 00m 00s
	Days Of Week	✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true	
Capacity	Unlimited	
Full Policy	Roll	
Time Zone	EST (-5)	
Local History Name Format	%name%	

This dialog shows *configuration* properties of the history import descriptor, plus Name (equivalent to the right-click Rename command on the descriptor). To access *all properties*, (including all status properties) go to its *property sheet*.

For related property details, see "BACnet Trend Log notes" in the *BACnet Guide*, or if a Niagara History Import descriptor, see "Niagara History Import properties" on page 2-31.

The following related topics also apply:

- History descriptor "gang" edits
- "Manager table features" on page 1-21

History descriptor "gang" edits The Edit button in the History Import (or Export) Manager allows you to edit one or more descriptors in a single dialog. Before clicking Edit, use standard Windows keyboard controls to highlight (select) *multiple descriptors* (e.g. hold down Ctrl key and click desired descriptors).

Note: *Edit is also on the History Import (or Export) Manager toolbar and the Manager menu, if any descriptor is selected.*

The "gang" edit feature is useful for making identical changes in multiple (selected) descriptors. For example, you can change "Execution Time" (when data is pulled into imported history) in multiple descriptors at the same time (versus editing individually).

When you have the Edit dialog open with multiple descriptors, you can also click on select ones to make individual edits (e.g. Execution, Time of Day), during the same dialog session where you made “gang” edits. When you click OK, all the changes are applied to the descriptors as made during your Edit session.

Note: *When you have multiple descriptors selected in the Edit dialog, properties that currently have different values are automatically unavailable (dimmed). Only a property that currently has the same value (across all selected descriptors) is available for gang edit.*

About History Import Discover, Add and Match (Learn Process)

Unless working offline, you can use the learn process to import histories in the station. As with other NiagaraAX learns, this is a two-step *process* in the History Import Manager, where you:

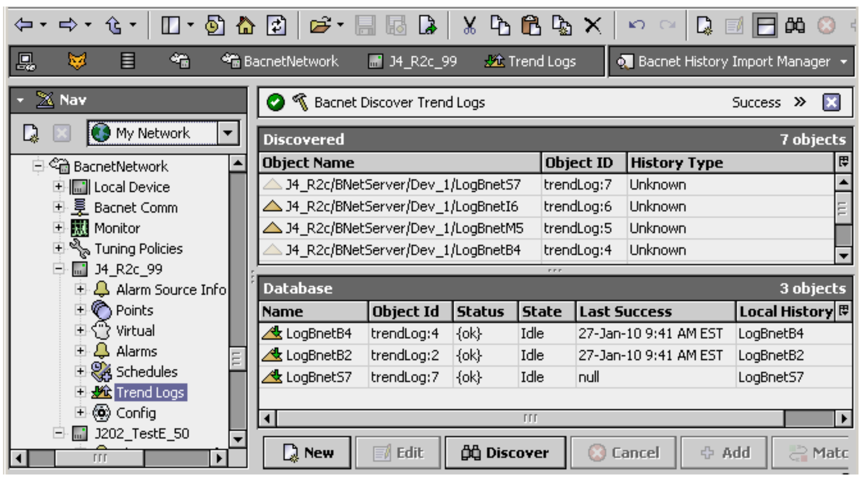
1. Under a selected device component, use its (Histories extension) Histories Import Manager view to Discover log data (histories) as *candidates* for inclusion in the station as histories.
2. Select and Add from those histories, creating history descriptors under the device’s Histories container.

Note: *The Histories Import Manager reinforces this process by providing two separate panes in the view whenever you enter “Learn Mode.” See “About Learn toggle” on page 1-18.*

Discover When you click Discover, the Histories Import Manager splits into two panes (*Learn Mode*): discovered items in the top pane, and existing import descriptors bottom pane (Figure 1-62).

Note: *If under a BacnetDevice, a “Bacnet Trend Logs Discover” job is started, with a progress bar at top. If BACnet Trend Log objects are found, they are listed in the discovered pane.*

Figure 1-62 *Discover splits History Import Manager (Bacnet History Import Manager shown)*



Discovered 7 objects

Object Name	Object ID	History Type
J4_R2c/BNetServer/Dev_1/LogBnetS7	trendLog:7	Unknown
J4_R2c/BNetServer/Dev_1/LogBnetI6	trendLog:6	Unknown
J4_R2c/BNetServer/Dev_1/LogBnetM5	trendLog:5	Unknown
J4_R2c/BNetServer/Dev_1/LogBnetB4	trendLog:4	Unknown

Database 3 objects

Name	Object Id	Status	State	Last Success	Local History
LogBnetB4	trendLog:4	{ok}	Idle	27-Jan-10 9:41 AM EST	LogBnetB4
LogBnetB2	trendLog:2	{ok}	Idle	27-Jan-10 9:41 AM EST	LogBnetB2
LogBnetS7	trendLog:7	{ok}	Idle	null	LogBnetS7

Note: *Under a NiagaraNetwork (only), the discovered pane shows the collapsed tree structure of all Niagara histories of that selected NiagaraStation. Click to expand and select histories for import. See “Discovered selection notes” on page 2-31 for more details.*

In Learn Mode, the two panes in the Histories Import Manager operate as follows:

- **Discovered (top pane)**
  Lists histories (Niagara) or Trend Logs (Bacnet) found in the device, as history descriptor candidates. Any item that already *exists as a history* in the station is “ghosted” (faintly listed).
  If a BacnetDevice, a job “progress bar” is also included on top of the Discovered pane.
  Note: *A Cancel button is available during a discovery job. If needed, use it to stop discovery.*
- **Database (bottom pane)**
  Lists history descriptors currently in the station database (each has an associated history).

Note: *As necessary, drag the border between the two panes to resize. Also (at any time), toggle between the two-pane Learn Mode and the single-pane (Database) view by clicking the Learn Mode tool in the toolbar (Figure 1-24 on page 18), or using the Learn Mode command in the Manager menu.*

Add The Add button is available in Learn Mode when you have one or more items selected (highlighted) in the top discovered pane. When you click Add, a dialog appears that allows you to edit properties before the history descriptor(s) are created in the station.

Note: *Whenever you select one or more items in the top discovered pane, the toolbar also has an available Add tool (“plus” symbol), and the Manager menu has an Add command. Also, you can simply double-click a discovered item to bring it up in the Add dialog.*

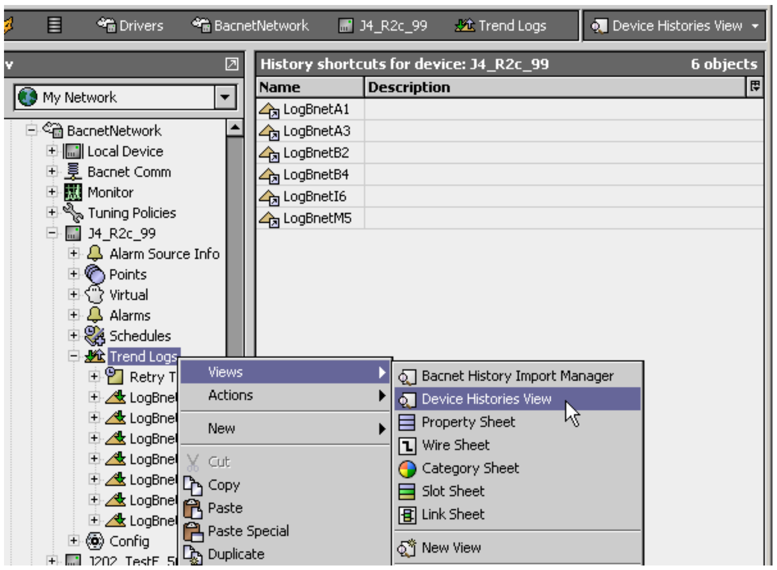
The Add dialog is identical to the history import descriptor Edit dialog. For details on properties of a *Niagara* HistoryImport descriptor, see "Niagara History Import properties" on page 2-31.

Match Match, as an online function, is available if you have *one* history selected in the top (discovered) pane and *one* history import descriptor selected in the bottom (database) pane. However, usage of Match when importing histories from a device (NiagaraStation, BacnetDevice) is generally *not* recommended. Instead, use the Discover and Add method to import histories.

About the Device Histories View

Starting in AX-3.5, 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 1-63 *Device Histories View on the Histories (Trend Logs) extension of a BacnetDevice component*



History shortcuts for device: J4_R2c_99 — 6 objects

Name	Description
LogBnetA1	
LogBnetA3	
LogBnetB2	
LogBnetB4	
LogBnetI6	
LogBnetM5	

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 Niagara 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. For related details, see "About history nav shortcuts" in the *User Guide*.

About Schedules extension views

For a Schedules device extension overview, see "About the Schedules extension" on page 1-36.

Special views on a device's Schedules extension may include:

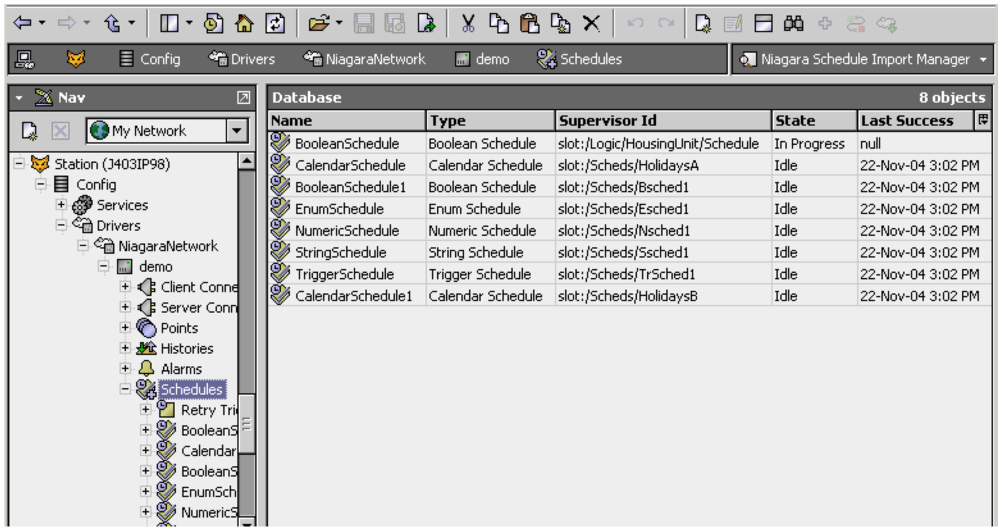
- Schedule Import Manager
- Schedule Export Manager

Note: *Other standard views of a Schedules extension are commonly used. For example, the wire sheet may be used when linking imported schedules to other control logic in the station.*

Schedule Import Manager

The Schedule Import Manager is the *default view* for a device's *Schedules* extension. Like other managers, it is a table-based view (Figure 1-64). Each row corresponds to an *imported Niagara schedule* (read-only). Configuration for each includes its name and whether it is enabled.

Figure 1-64 *Schedule Import Manager under a NiagaraStation*



Name	Type	Supervisor Id	State	Last Success
BooleanSchedule	Boolean Schedule	slot:/Logic/HousingUnit/Schedule	In Progress	null
CalendarSchedule	Calendar Schedule	slot:/Scheds/HolidaysA	Idle	22-Nov-04 3:02 PM
BooleanSchedule1	Boolean Schedule	slot:/Scheds/Bsched1	Idle	22-Nov-04 3:02 PM
EnumSchedule	Enum Schedule	slot:/Scheds/Esched1	Idle	22-Nov-04 3:02 PM
NumericSchedule	Numeric Schedule	slot:/Scheds/Nsched1	Idle	22-Nov-04 3:02 PM
StringSchedule	String Schedule	slot:/Scheds/Ssched1	Idle	22-Nov-04 3:02 PM
TriggerSchedule	Trigger Schedule	slot:/Scheds/TrSched1	Idle	22-Nov-04 3:02 PM
CalendarSchedule1	Calendar Schedule	slot:/Scheds/HolidaysB	Idle	22-Nov-04 3:02 PM

When building a network in the station, you use this view to create, edit, and delete imported Niagara schedules. In the case of a Niagara network (only), each schedule that you import results in the creation of a remote "schedule export descriptor" in that remote Niagara 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" on page 1-34), or if a NiagaraStation, schedule export descriptors. However, the Nav tree and other views on Schedules provide you access to these items.*

At the bottom of the view, the button "New" lets you manually create new imported schedules in the station. An "Edit" button lets you edit a few properties of one or more imported schedules. Buttons "Discover," "Add" and "Match" are also available, (these work similarly as in the Point Manager). An "Import" button is available to manually import (pull data) into one or more selected imported schedules. Finally, depending on driver, additional buttons may be available.

For more details, see:

- Schedule Import New
- Schedule Import Edit
- About Schedule Import Discover, Add and Match (Learn Process)

Schedule Import New

Button New exists in a device's Schedule Import Manager, but 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).

Note: *A "New" tool is also on the Schedules Import Manager toolbar, and in the Manager menu.*

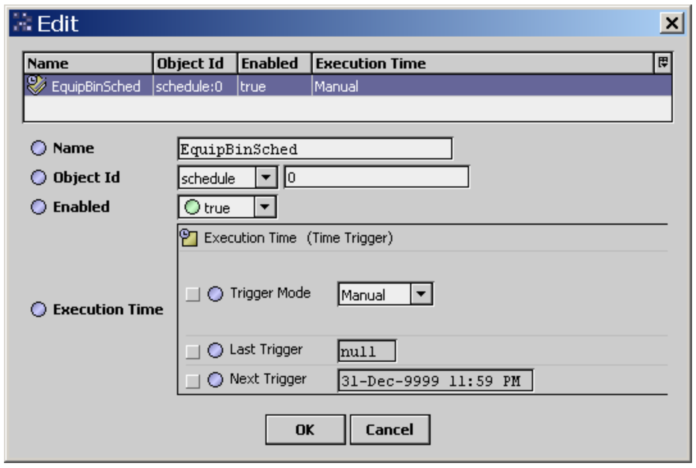
Schedule Import Edit

In the Schedule Import Manager, you can Edit selected properties of a schedule import extension in the station database by simply double-clicking it.

Edit

The Edit dialog appears with the imported schedule import descriptor listed (Figure 1-65).

Figure 1-65 *Edit dialog in Schedule Import Manager (Bacnet)*



Name	Object Id	Enabled	Execution Time
EquipBinSched	schedule:0	true	Manual

Name: EquipBinSched
Object Id: schedule 0
Enabled: true
Execution Time: Execution Time (Time Trigger)
Trigger Mode: Manual
Last Trigger: null
Next Trigger: 31-Dec-9999 11:59 PM

Note: *The Edit dialog shows some properties of the schedule's ScheduleImportExt, plus Name—equivalent to the right-click Rename command on the parent schedule component). To access all properties of the schedule (including all status properties) go to its property sheet, or to see the imported schedule events, go to its Scheduler view. Or, in the Nav tree you can simply double-click the schedule for its Scheduler view.*

The following related topics also apply:

- Schedule Import properties
- Schedule Import or Export "gang" edits
- Manager table features

Schedule Import properties

Properties of imported schedules available in the Edit or Add dialog of the Schedule Import Manager are as follows:

- **Name**
  Name for the imported Niagara schedule component. If discovered, will match the name of the source schedule. Must be unique among other components in same container.
  Note: *Editing name does not affect name of the source schedule, nor the name of the corresponding schedule export descriptor (if source is a Niagara schedule).*
- **Supervisor Id**
  (NiagaraStation only) Unique slot path of source Niagara schedule in that station.
- **Object Id**
  (BacnetDevice only) Combination of BACnet object type (schedule or calendar) and instance number (unique within that object type in that device).
- **Enabled**
  Default is true. If set to false, the imported schedule is disabled.
  Note: *While disabled, the schedule's Out slot has status disabled. Any downstream logic linked to Out no longer processes it. Refer to the section "About "isValid" status check" in the* User Guide.
- **Execution Time**
  (BacnetDevice only) Specifies how event-configuration refresh (import) occurs with source schedule, using a "pull" request method. Options are Daily, Interval, or Manual (default).
  If Manual, some properties below are not available, as noted:
  - **Time of Day (Daily)**
    Configurable to any daily time. Default is 2:00am.
  - **Randomization (Daily)**
    When the next execution time calculates, a random amount of time between zero milliseconds and this interval is added to the Time of Day. May prevent "server flood" issues if too many schedule imports execute at the same time. Default is zero (no randomization).
  - **Days of Week (Daily and Interval)**
    Select (check) days of week for import execution. Default is all days of week.
  - **Interval (Interval)**
    Specifies repeating interval for import execution. Default is every 15 minutes.
  - **Time of Day (Interval)**
    Specifies start and end times for interval. Default is 24-hours (start 12:00am, end 11:59pm).

- **Last Trigger**
  Timestamp of when last interval or daily import occurred.
- **Next Trigger**
  Timestamp of when the next interval or daily import is configured to occur.

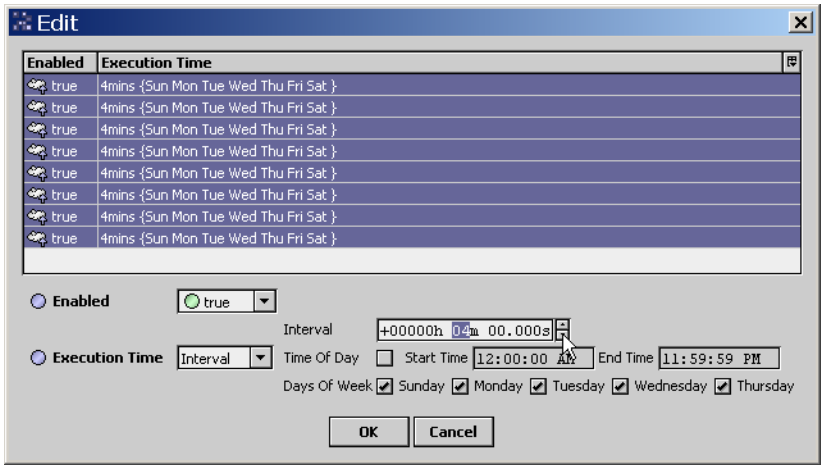
Schedule Import or Export "gang" edits

The Edit button in the Schedule Import (or Export) Manager allows you to edit one or more items in a single dialog. Before clicking Edit, use standard Windows keyboard controls to highlight (select) *multiple items* (e.g. hold down Ctrl key and click desired schedules).

Note: *Edit is also on the Schedule Import (or Export) Manager toolbar and the Manager menu, if any item in the database is selected.*

The "gang" edit feature is useful for making identical changes in multiple (selected) items. For example, as shown in Figure 1-66 when using Edit in the Niagara Schedule Export Manager, you can change the "Execution Time, Interval" in multiple schedule export descriptors at the same time (versus editing individually). For more details, see "About gang edits."

Figure 1-66 *Example "gang edit" in Edit dialog of Schedule Export Manager*



Enabled	Execution Time
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }
true	4mins {Sun Mon Tue Wed Thu Fri Sat }

Enabled: true
Execution Time: Interval
Interval: +00000h 04m 00.000s
Time Of Day ☐ Start Time 12:00:00 AM End Time 11:59:59 PM
Days Of Week ✓ Sunday ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday

About Schedule Import Discover, Add and Match (Learn Process)

Unless working offline, you can use the learn process to import schedules in the station. As with other NiagaraAX learns, this is a two-step *process* in the Schedule Import Manager, where you:

1. Under a selected device component, use its (Schedules extension) Schedule Import Manager view to Discover schedules as *candidates* for inclusion in the station as Niagara schedules.
2. Select and Add from those candidates, creating imported schedules under the device's Schedules container.

Note: *The Schedule Import Manager reinforces this process by providing two separate panes in the view whenever you enter "Learn Mode." See "About Learn toggle" on page 1-18.*

Discover

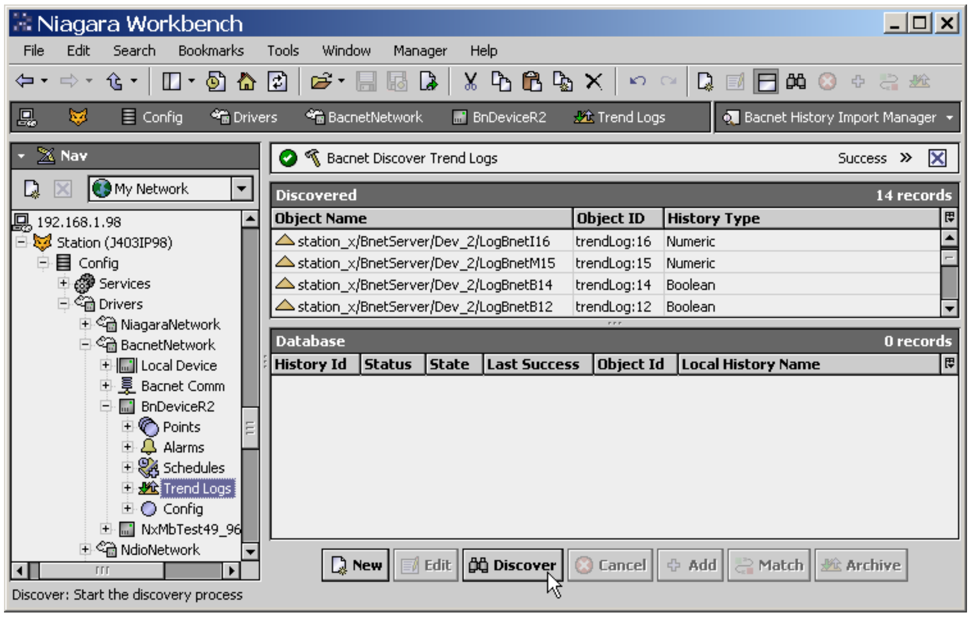
When you click Discover, the Schedule Import Manager splits into two panes (*Learn Mode*): discovered items in top pane, and existing imported schedules in bottom pane (Figure 1-67).

A schedule discovery job (either Niagara or Bacnet) is started, with a progress bar at top.

- If Niagara schedules are found in the remote station, they are listed in the discovered pane.
- If BACnet Schedule and/or Calendar objects are found, they are listed in the discovered pane.

Note: *A Cancel button is available during a discovery job. If needed, use it to stop discovery.*

Figure 1-67 *Discover splits Schedules Import Manager*



Bacnet Discover Trend Logs Success

Discovered 14 records

Object Name	Object ID	History Type
station_x/BnetServer/Dev_2/LogBnetI16	trendLog:16	Numeric
station_x/BnetServer/Dev_2/LogBnetM15	trendLog:15	Numeric
station_x/BnetServer/Dev_2/LogBnetB14	trendLog:14	Boolean
station_x/BnetServer/Dev_2/LogBnetB12	trendLog:12	Boolean

Database 0 records

History Id	Status	State	Last Success	Object Id	Local History Name

Discover: Start the discovery process

In Learn Mode, the two panes in the Schedule Import Manager operate as follows:

- **Discovered (top pane)**
  Lists schedule components (Niagara) or Schedule and/or Calendar objects (Bacnet) found in the device, as candidates for imported schedules. Any item that already *exists as a schedule* in the station is "ghosted" (faintly listed).
- **Database (bottom pane)**
  Lists schedules currently imported in the station database (contained in Schedules container).

Note: *As necessary, drag the border between the two panes to resize. Also (at any time), toggle between the two-pane Learn Mode and the single-pane (Database) view by clicking the Learn Mode tool in the toolbar (Figure 1-24 on page 18), or using the Learn Mode command in the Manager menu.*

Add

The Add button is available in Learn Mode when you have one or more items selected (highlighted) in the top discovered pane. When you click Add, a dialog allows you to edit properties *before* the schedule is created in the station. The Add dialog and Edit dialog are identical.

Note: *Whenever you select one or more items in the top discovered pane, the toolbar also has an available Add tool ("plus" symbol), and the Manager menu has an Add command. Also, you can simply double-click a discovered item to bring it up in the Add dialog.*

For details on properties, see "Schedule Import properties" on page 1-51.

Match

Match, as an online function, is available if you have *one* schedule selected in the top (discovered) pane and *one* schedule selected in the bottom (database) pane. However, usage of Match when importing schedules from a device (NiagaraStation, BacnetDevice) is generally *not* recommended. Instead, use the Discover and Add method to import schedules.

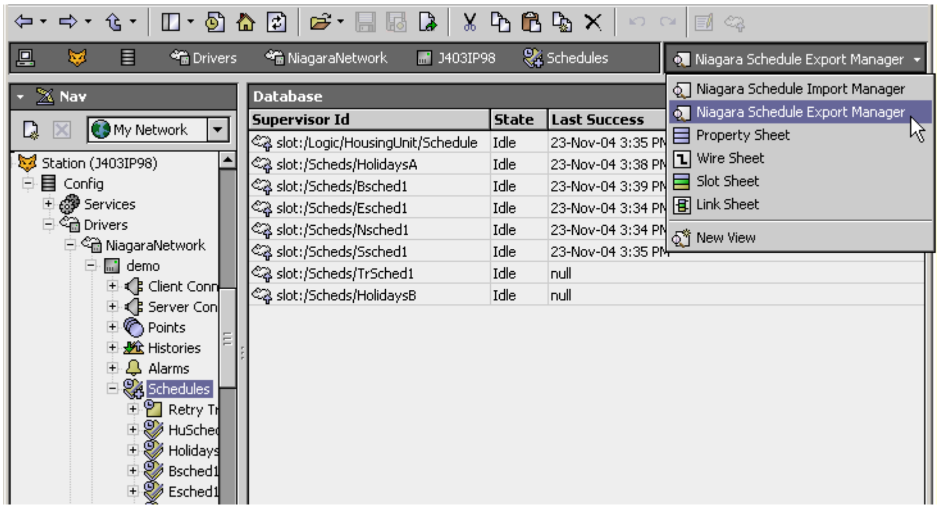
Schedule Export Manager

The Schedules extension of a NiagaraStation, ObixClient/R2ObixClient, or BacnetDevice has an available Export Manager view. It allows management of schedule components in the local station made available to that remote device.

Like other managers, the Schedule Export Manager is a table-based view (Figure 1-68). 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 1-68 *Schedule Export Manager under a NiagaraStation*



Supervisor Id	State	Last Success
slot:/Logic/HousingUnit/Schedule	Idle	23-Nov-04 3:35 P
slot:/Scheds/HolidaysA	Idle	23-Nov-04 3:38 P
slot:/Scheds/Bsched1	Idle	23-Nov-04 3:39 P
slot:/Scheds/Esched1	Idle	23-Nov-04 3:34 P
slot:/Scheds/Nsched1	Idle	23-Nov-04 3:34 P
slot:/Scheds/Ssched1	Idle	23-Nov-04 3:35 P
slot:/Scheds/TrSched1	Idle	null
slot:/Scheds/HolidaysB	Idle	null

Note: *A Schedule Export Manager works differently in NiagaraNetworks and BacnetNetworks/ObixNetworks.*

- For 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 upon the remote Niagara station “importing” a local schedule component. For more details, see “Station Schedules import/export notes” on page 2-28.
- 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. For more details, see the NiagaraAX BACnet Guide or NiagaraAX oBIX Guide.

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 (see “About the Retry Trigger” on page 1-34). However, the Nav tree and other views on Schedules provide you access to these items.*

CHAPTER 2

Niagara Network

About the Niagara Network

Currently, by default *every* NiagaraAX station has a Niagara Network under its Drivers container. The Niagara Network is where data is modeled that *originates from other stations*. Generally, this is done using the same driver architecture used by other (non-Niagara) drivers. See "About Network architecture" on page 1-2.

A Niagara Network has the following unique differences:

- All proxy points under a NiagaraStation are "read only" types, namely one of the following:
  - **BooleanPoint**
  - **EnumPoint**
  - **NumericPoint**
  - **StringPoint**

  However, note that a Niagara proxy point inherits any *actions* from the originating (remote) point or object. For more details, see "Niagara proxy point notes" on page 2-24.
- 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. A Workbench connection to a station operates identically, where Workbench is the client, and station is server. Client authentication (performed by the server) is required in all Fox connections.

Note: *Starting in AX-3.7, the Fox protocol can run over an SSL encrypted connection following certificate-based server authentication. This secure Fox SSL (or Foxs) is noted and mentioned in various subsections of this document update. For complete details, refer to the* NiagaraAX SSL Connectivity Guide*.*

For more details specific to a Niagara Network, see the following main sections:

- NiagaraNetwork component notes
- Niagara Station Manager notes
- NiagaraStation component notes
- About the Users extension
- Niagara Point Manager notes
- Niagara proxy point notes
- NiagaraStation Alarms notes
- Station Schedules import/export notes
- About Niagara virtual components

NiagaraNetwork component notes

In addition to common network components (see "Common network components" on page 1-5), the Niagara Network contains components specific to Niagara, namely:

- **Local Station**
  A container of read-only "Sys Def" properties that reflect what information would be "sync'ed up" to a remote Supervisor station (if the local station was defined as a subordinate). Sys Def is potentially of use to developers. For details, "About Sys Def components" on page 2-51.
- **Sys Def Provider**
  A container for "Sys Def" child components, of type "ProviderStations", which are typically hidden slots (by default). The API interacts with this component to query about the "Sys Def" hierarchy, and persists this definition. Sys Def is chiefly of interest to NiagaraAX developers working with the API. For details, "About Sys Def components" on page 2-51.

- **Fox Service**
  A container for Fox protocol settings affecting client connections made to the local station, such as from Workbench or from another station. Such connections appear locally as "server connections." For details, see "About the Fox Service" on page 2-2.
- **History Policies**
  A container for "rules" that specify how remotely-generated histories should be changed when these histories are pushed into the station (that is, *exported* from remote stations). Also contains a poll scheduler for "on demand" polling of histories. For details, see "About History Policies" on page 2-7.
- **Workers**
  A container with a configurable property "Max Threads". This property allows tuning for large Niagara networks (many NiagaraStation components), and is the only visible part of a "shared thread pool scheme" for large-job scalablilty. In current releases, station work from a single station was limited to one thread, and the former default value of 50 was changed to a "`max`" value. This allows the local station's thread pool to grow uncapped while alleviating a former thread starving issue.
  Very large Supervisors with many stations may benefit from Niagara thread pool adjustments made via entries in the host's `system.properties` file. Consult your support channel for details.

In addition, tuning policies are simplified in a Niagara Network, as compared to "polling type" drivers. See the next "Niagara Tuning Policy notes" for more information.

Niagara Tuning Policy notes

Note that **`Tuning Policies`** for a NiagaraNetwork have only 3 properties, as follows:

- **Stale Time**
  - If set to a non-zero value, a subscribed Niagara 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 Niagara proxy point has a stale status, for example a Bacnet proxy point, the Niagara 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 Niagara 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**
  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**
  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.

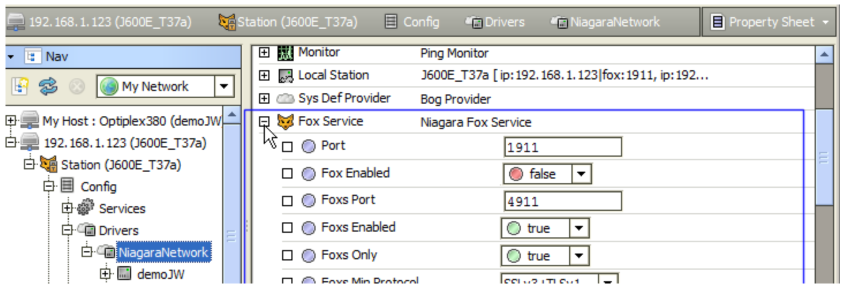
Note: *Relative to tuning policies in other networks, the importance of NiagaraNetwork tuning policies are typically secondary, and then only applicable for a station that has proxy points under its NiagaraNetwork. In typical applications, this means the Supervisor station only.*

As a general rule, if configuring the Stale Time in a Niagara Tuning Policy, it is recommended to be greater than the Max Update Time by a factor of three.

About the Fox Service

The NiagaraNetwork's Fox Service holds configuration for the *local* station's Fox settings. (This varies from most other services, which are found instead under the station's **`ServiceContainer`**.)

Figure 2-1 *FoxService in property sheet of NiagaraNetwork*



Monitor	Ping Monitor
Local Station	J600E_T37a [ip:192.168.1.123|fox:1911, ip:192...
Sys Def Provider	Bog Provider
Fox Service	Niagara Fox Service
Port	1911
Fox Enabled	🔴 false
Foxs Port	4911
Foxs Enabled	🟢 true
Foxs Only	🟢 true
Foxs Min Protocol	

Included are properties for the TCP port number assigned to the Fox server, authentication method used, and various timeout/trace settings. If the host platform is configured for SSL (recommended if AX-3.7 or later), often you change a few related properties from defaults. See Fox Service properties for more details.

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

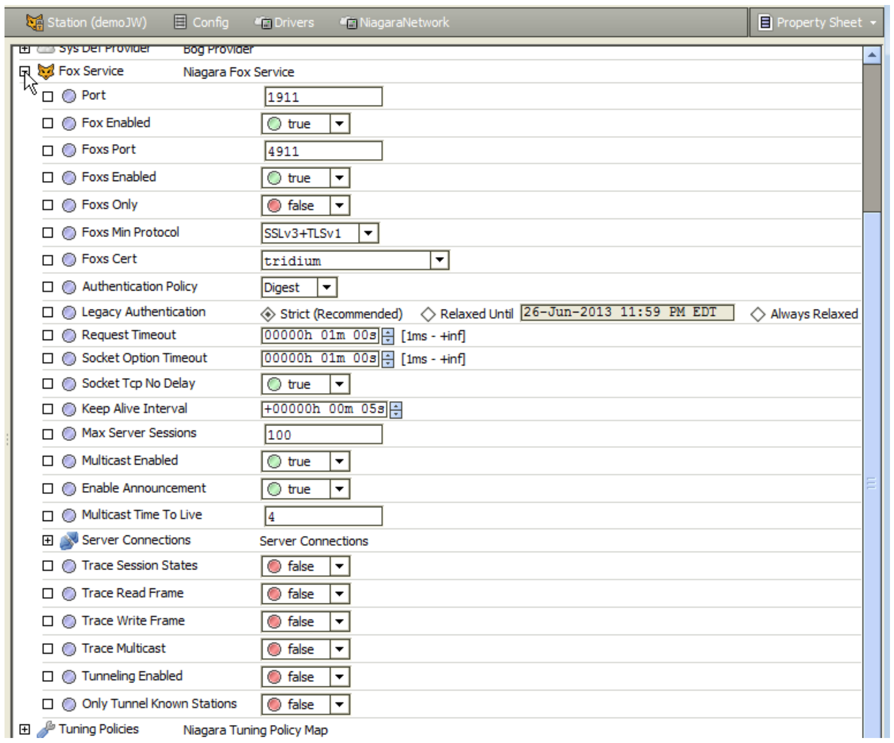
- If opening a station in Workbench, you must enter a valid station username and password for it in the station login dialog (otherwise it does not open).
- If accessing a station in a browser as a user, where you also must enter valid user credentials (log in).
- If adding a NiagaraStation to a station's 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. For related details, refer to the section "Multi-station security notes" in the* User Guide.

The Fox Service also has a Server Connections container slot with a default ServerConnectionsSummary view. Client connections to the station's Fox server are dynamically modeled as "Session*N*" child components. The summary view allows you to see all current connections, and if necessary, perform a right-click `Force Disconnect` action on one or more connected users.

Fox Service properties

Figure 2-2 shows the Fox Service expanded in the NiagaraNetwork property sheet of an AX-3.7u1 or later station.

Figure 2-2 *Fox Service properties in property sheet of an AX-3.7 NiagaraNetwork*



Sys Def Provider	Bog Provider
Fox Service	Niagara Fox Service
Port	1911
Fox Enabled	true
Foxs Port	4911
Foxs Enabled	true
Foxs Only	false
Foxs Min Protocol	SSLv3+TLSv1
Foxs Cert	tridium
Authentication Policy	Digest
Legacy Authentication	[✓ Strict (Recommended)] Relaxed Until 26-Jun-2013 11:59 PM EDT Always Relaxed
Request Timeout	00000h 01m 00s [1ms - +inf]
Socket Option Timeout	00000h 01m 00s [1ms - +inf]
Socket Tcp No Delay	true
Keep Alive Interval	+00000h 00m 05s
Max Server Sessions	100
Multicast Enabled	true
Enable Announcement	true
Multicast Time To Live	4
Server Connections	Server Connections
Trace Session States	false
Trace Read Frame	false
Trace Write Frame	false
Trace Multicast	false
Tunneling Enabled	false
Only Tunnel Known Stations	false
Tuning Policies	Niagara Tuning Policy Map

Fox Service properties are described as follows:

- **Port**
  Specifies the TCP port used by Fox server (default is 1911).
- **Fox Enabled**
  Default (before AX-3.8) is true. Must be true for Fox communications to occur on the port above.
  Note: *If station communications are needed between any other NiagaraAX host that is not configured or capable of AX-3.7 or later SSL (such as any JACE-2), set this at* true. *Note that in AX-3.8, a "new station" using defaults has this property set to* `false`. *See "FoxService defaults (new station) changed in AX-3.8" on page 2-6.*

- **Foxs Port**
  Specifies the TCP port used by Foxs (secure socket-layer) server, where default is 4911.
  Note: *This and other Foxs properties below apply only if the station's host is configured and licensed for SSL/TLS, including installed certificate(s).*
- **Foxs Enabled**
  Default (before AX-3.8) is false. Must be true for Foxs communications to occur on the port above.
  Note: *In AX-3.8, a "new station" using defaults has this property set to* `true`*. See "FoxService defaults (new station) changed in AX-3.8" on page 2-6.*
- **Foxs Only**
  Default (before AX-3.8) is false. If true, and Fox Enabled and Foxs Enabled are both true, Fox connection attempts are *redirected* as Foxs connections. Such a configuration is different from setting Fox Enabled false and Foxs true. In that case, only Foxs connection attempts work; Fox connection attempts are ignored.
  Note: *If station communications are needed between any other NiagaraAX host that is not configured or capable of AX-3.7 or later SSL (such as a JACE-2), set this at* false*. Note in AX-3.8, a "new station" using defaults has this property set to* `true`*. See "FoxService defaults (new station) changed in AX-3.8" on page 2-6.*
- **Foxs Min Protocol**
  Specifies which standard protocol the Foxs server supports for client connections, where the default is SSLv3+TLSv1 (both, the default). Other choices are SSLv3 (only) or TLSv1 (only).
- **Foxs Certificate**
  The "alias" for the server certificate in the host platform's "key store" to use for any Foxs connection. The default is the "tridium" self-signed certificate, which is automatically created when SSL is first loaded (by presence of certain modules and proper host licensing). If other certificates are in the host platform's key store, you can use the drop-down control to select another one instead.
- **Authentication Policy**
  These policy options apply to how password authentication is handled by the default NiagaraAX User Service or any other service when establishing a client connection using the Fox protocol. Property choices (when using AX-3.7u1) are as follows:
  - **Digest**
    This option uses password hashing. This is the recommended setting when using the default User Service.
  - **Basic**
    This option does not use encryption, therefor the user password is passed in clear text. Typically, Basic is used only if using LDAP.

  Note: *Prior to AX-3.7u1 (in AX-3.7), the "Digest" choice was "Digest MD5", which equates to "legacy encryption". This was strengthened to SCRAM-SHA (Salted Challenge Response Authentication Mechanism) in AX-3.7u1 and later. Any station upgraded from AX-3.7 will automatically upgrade from "Digest MD5" to "Digest". This also relates to a new FoxService property, "Legacy Authentication", described below.*

  Additionally, a third "Transactional" choice for Authentication Policy (available in prior AX-3.7) is no longer available, as products that were designed to use it were discontinued.
- **Legacy Authentication**
  (AX-3.7u1 or later only) Applies to password authentication when establishing a connection with a remote Fox client (either another station *or Workbench*). Choices are as follows:
  - **Strict**
    (Recommended, and *default*) Remote client must be able to handle passwords using SCRAM-SHA, requiring it to be running AX-3.7u1 or later. Clients running earlier "pre-2013" NiagaraAX releases are considered "legacy clients", and will be unable to connect.
  - **Relaxed Until (date)**
    Allows a legacy client to connect to this station using Basic authentication, but only until the specified date. After that date, this setting behaves the same as if set to Strict.
  - **Always Relaxed**
    Always allows a legacy client to connect to this station using Basic authentication. Be especially careful if setting to this value—see the Note and Caution below!

  Note: *Both "Relaxed" choices are intended for* temporary usage only *during a system upgrade to one of the "security update" releases. For example, after upgrading a Supervisor, you could set its Legacy Authentication to relaxed only until all its subordinate JACEs have also been upgraded. This would allow those stations to continue to communicate to the Supervisor until upgraded. Strict is the most secure option. After a system-wide upgrade, set this to Strict in the* FoxService of all stations!

For related details, refer to "Upgrade considerations" in NiagaraAX 2013 Security Updates*.*

Caution *Unless set to "Strict", either "Relaxed" option is effectively the same as setting the FoxService property* Authentication Policy *to "Basic" for any legacy connection attempt, as this is the only "common ground" authentication between a legacy Fox client (Workbench or another station) and this update release station. Thus, passwords for client connections to this station are passed in clear text (if a legacy client). Obviously, this is not a desired operating state. For this reason, you should seldom (if ever) set the* Legacy Authentication *property to "Always Relaxed".*

- **Request Timeout**
  Time to wait for a response before assuming a connection is dead (default is 1minute).
- **Socket Option Timeout**
  Time to wait on a socket read before assuming the connection is dead (default is 1minute).
- **Socket Tcp No Delay**
  Used to disable "Nagle's algorithm", found to cause issues with delayed acknowledgements that occurred in TCP socket communications between Fox clients and servers. The default (and typically recommended) value is `true`, disabling Nagle's algorithm.
  Note: *On the Workbench side, there is a line in the* `system.properties` *file you can enter to adjust this setting: "*`niagara.fox.tcpNoDelay=true`*".*
- **Keep Alive Interval**
  Interval between keep alive messages (default is 5 seconds). The keep alive should be well below the request timeout and socket option timeout.
- **Max Server Sessions**
  Maximum number of Fox/Foxs server connections before client connections error with busy. Default is 100.
- **Multicast Enabled**
  Default is true. Allows UDP multicasting *initiated* by the station, necessary for a discovery *from* this station. Note this is different than Workbench UDP mulitcast support, which can be optionally disabled via an entry in the Workbench host's `system.properties` file.
- **Enable Announcement**
  Either true (default) or false. Enables support of UDP multicast announcement messages *received* by the station, in support of learn/discover.
- **Multicast Time To Live**
  Number of hops to make before a multicast message expires (default is 4).
- **Server Connections**
  Provides status information about current Workbench client connections to the local station (does not reflect station-to-station Fox connections).
- **Trace Session States**
  Either true or false (default). Debug usage for tracing session state changes.
- **Trace Read Frame**
  Either true or false (default). Debug usage for dumping frames being read from the wire.
- **Trace Write Frame**
  Either true or false (default). Debug usage for dumping frames being written to the wire.
- **Trace Multicast**
  Either true or false (default). Debug usage for tracing multicast messaging.
- **Tunneling Enabled**
  If enabled, allows the station to perform as a Fox "tunnel" (i.e. proxy server) through which Workbench sessions can be established to other Niagara hosts, typically unreachable otherwise. A special tunnel ORD syntax is used from the client Workbench to reach target stations through the Fox tunnel. Usage requires the proxy server (typically the Supervisor host) to be licensed for Fox tunneling. For more details, see "About Fox Tunneling" in the engineering notes document *Fox Tunneling and HTTP Tunneling*. Note this property *does not affect platform tunneling*, introduced in AX-3.5.
- **Only Tunnel Known Stations**
  An added security option that affects Fox tunneling, HTTP tunneling, and platform tunneling. Applies only if the station is configured as a tunnel (proxy server).
  - If left at the default (false) value, a tunnel connection is permitted to *any* target IP address given in the tunnel ORD, or in the case of platform tunneling, to any "Tunnel to" IP address entered in the Workbench "Open Platform" dialog.
  - If set to true, Fox, HTTP, and platform tunneling is permitted *only* to a station that exists in the proxy server's NiagaraNetwork, where tunnel ORD (Fox) or URL (HTTP) syntax uses the target station's *name* instead of IP address. For platform tunneling, the target "Tunnel to" in the

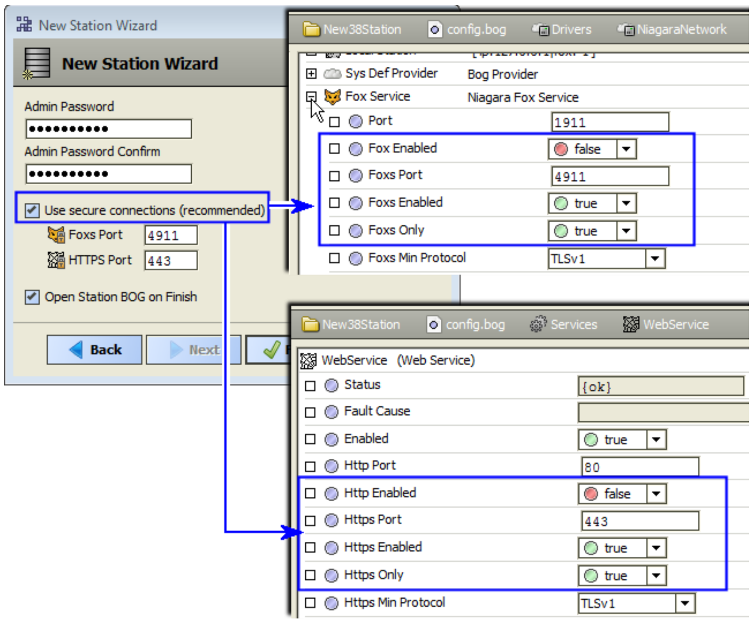
Workbench "Open Platform" dialog also requires the *station name*.
For example, going through 10.10.8.9 to reach target station "demoStation" at 10.10.5.4
– *instead* of (Fox) tunnel at `ip:10.10.8.9|fox:/10.10.5.4`
– *now* (Fox) tunnel at `ip:10.10.8.9|fox:/demoStation`
or
– *instead* of (HTTP) tunnel at `http://10.10.8.9/tunnel/10.10.5.4`
– *now* (HTTP) tunnel at `http://10.10.8.9/tunnel/demoStation`
or
– *instead* of platform tunnel at "Host" = `10.10.8.9`, "Tunnel to" = `10.10.5.4`
– *now* platform tunnel at "Host" = `10.10.8.9`, "Tunnel to" = `demoStation`

Again, note that if this property is set to true, that the target NiagaraStation *name* must be used in the Fox tunnel ORD, HTTP URL, or Workbench "Open Platform" dialog—and not IP address.

FoxService defaults (new station) changed in AX-3.8

To encourage stronger security, any new station made using defaults from the **`New Station`** wizard in AX-3.8 Workbench is created with both its **`FoxService`** and **`WebService`** configured for "*SSL only*", or more accurately, with selected properties set as shown in Figure 2-3.

Figure 2-3 *FoxService and WebService "New station" defaults in AX-3.8*



Admin Password	••••••••••
Admin Password Confirm	••••••••••
☑ Use secure connections (recommended)	
Foxs Port	4911
HTTPS Port	443
☑ Open Station BOG on Finish	

Sys Def Provider	Bog Provider
Fox Service	Niagara Fox Service
Port	1911
Fox Enabled	false
Foxs Port	4911
Foxs Enabled	true
Foxs Only	true
Foxs Min Protocol	TLSv1

WebService (Web Service)	
Status	{ok}
Fault Cause	
Enabled	true
Http Port	80
Http Enabled	false
Https Port	443
Https Enabled	true
Https Only	true
Https Min Protocol	TLSv1

As shown above when using the **`New Station Wizard`**, if needed you can *change* the default "Foxs Port" and "Https Port" values from defaults (4911 and 443, respectively). Or, you can simply clear the default "Use secure connections" checkbox.

If you do the latter, the dialog changes to let you specify the (non-SSL) "Fox Port" and "Http Port" from defaults if needed (1911 and 80, respectively). Then the new station created has all Boolean properties above (circled in Figure 2-3) *reversed* for "non-SSL" usage: Fox Enabled=`true`, Foxs Enabled=`false`, and Foxs Only=`false`; in the WebService: Http Enabled=`true`, Https Enabled=`false`, Https Only=`false`. Essentially, this is how the **`New Station`** wizard works in AX-3.7u1 and prior releases.

In some cases, such as when making a station for a JACE-2/4/5 series controller, which does not support Fox SSL or Foxs (even if AX-3.7 or later), it makes sense to clear the "Use secure connections" option in the **`New Station`** wizard. Alternatively, you can simply edit the properties shown in Figure 2-3 as needed *after* the station is created, with its `config.bog` file opened in Workbench (offline).

For related details, see the sections "New Station wizard" and "WebService" in the NiagaraAX *User Guide*. For more details on SSL in NiagaraAX, refer to the *NiagaraAX SSL Connectivity Guide*.

About History Policies

The NiagaraNetwork's "`History Policies`" (History Network Ext) holds an "On Demand Poll Scheduler" that affects *imported* histories, if set up for "on demand polling". See "Default history policies".

History Policies also functions as the container for "Config Rules" that are used when remote histories are "exported" into the local station. Unlike *imported* histories, which let you define (and adjust later, if needed) the "capacity" and "full policy" settings in each HistoryImport descriptor, histories that are *exported into* the station have no associated component—only the history itself. The capacity and "full policy" for each history is set only *at creation time*, using the local history policies.

Note: *You export histories into the station working under a remote station—meaning, from a view of the Histories device extension under the NiagaraStation that represents this (local) station. See "Niagara History Export Manager" on page 2-33.*

For an example scenario showing histories in two JACE stations that are exported to a Supervisor station, and how history policies apply, see Figure 2-4 on page 8.

Additional details are in the following sections:

- Default history policies (including "On Demand Poll Scheduler")
- Config Rules (for histories *exported* into the local station)
- Example config rules

Default history policies

The following items are under a NiagaraNetwork's History Policies:

- **On Demand Poll Scheduler**
  Contains a set of standard poll component parameters which are listed and described in the section "Poll Service properties" on page 1-11. These parameters affect the operation of the "On Demand" polling for histories so enabled. For related details, see "On demand properties in history import descriptors" on page 2-32, and the *NiagaraAX User Guide* section "About On Demand history views".
- **Default Rule**
  This property has parameters—with wildcard matches to "all" stations and history names, specifying "unlimited" capacity and a "roll" fullPolicy. This means that *any* history that is exported into the station (from any remote station) is archived using a local history configured with *unlimited* capacity. Given the vast storage capacity of a supervisor host PC, the default "one rule" setting may be acceptable on the target of most exported histories (*supervisor station*). However, if for some reason you are exporting histories to a JACE station, you should definitely change the "Default Rule" of the History Policies under its NiagaraNetwork to specify smaller capacities. Even for a supervisor station, you may wish to change the default rule, and/or add additional optional config rules, as needed.

Config Rules

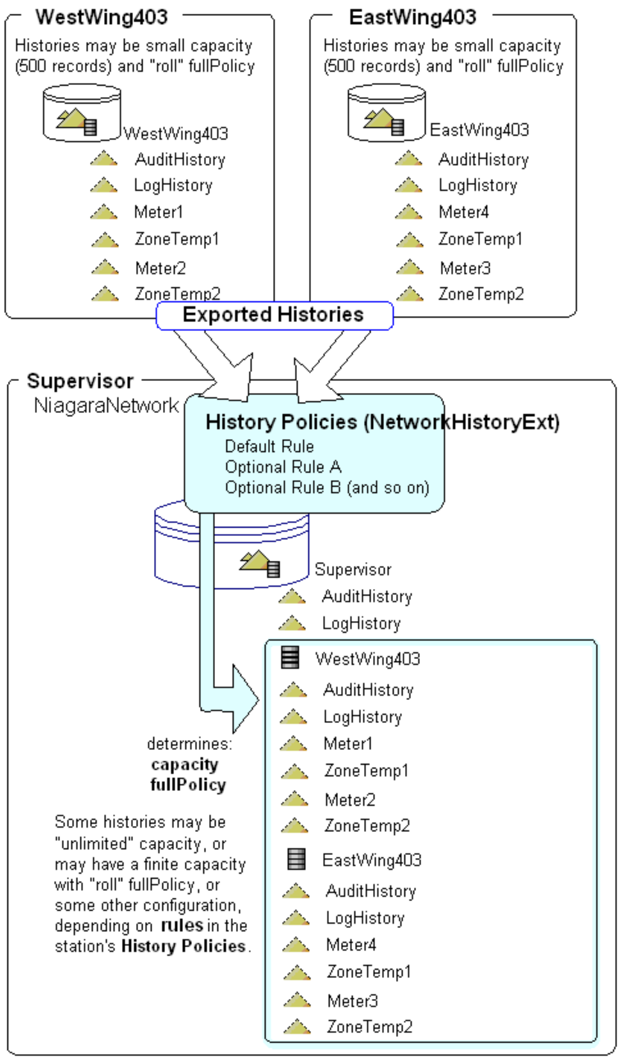
When a history is exported to the station (from another station), these rules are evaluated to set the local (archived) history's config properties "capacity" and "fullPolicy." The first matching rule is used. The "Default Rule" is always at top, and cannot be deleted or renamed.

Note: *Rule priority is set by order—as the "Default Rule" is always first, it is highest priority. If you create other rules (in Workbench, right-click a rule, then click "Duplicate"), you can edit, rename, and reorder as needed.*

Each config rule under the network's History Policies has the following configuration properties:

- **Device Pattern**
  String matching to device names, meaning name of NiagaraStation(s) that are exporting histories. Default value is a wildcard ("*"), meaning all station names are matched.
- **History Name Pattern**
  String matching to history names of histories being exported. Again, default value is a wildcard ("*"), meaning all named histories are matched.
  Note: *Both device pattern and history name pattern must apply for the rule to be used—otherwise the next rule down (in order) in History Policies is evaluated.*
- **capacity**
  The capacity setting to use when creating the local history, if matched by device and history names. Either "unlimited," or "Record Count," with a finite record specified.
- **fullPolicy**
  Applies if capacity is not "unlimited," and specifies if collection continues when the capacity record limit is reached ("roll") or collection stops ("stop").

Figure 2-4 *History Policies in a supervisor station*

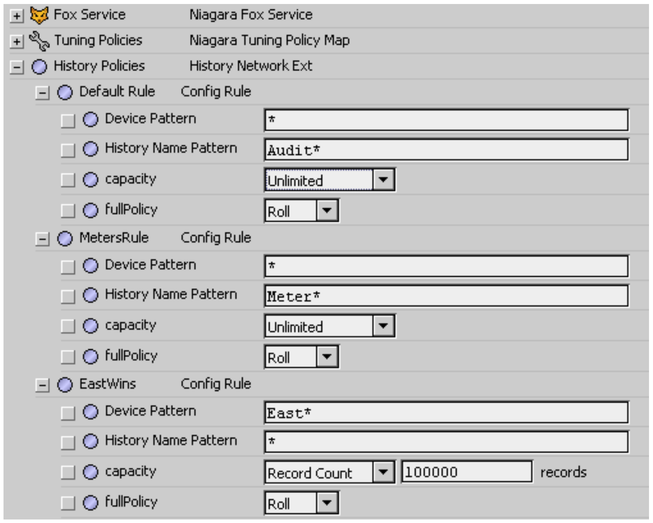


As shown in Figure 2-4, the Supervisor's History Policies "Config Rules" determine the capacity and fullPolicy settings for each history upon export from the JACE stations "WestWing403" and "EastWing403". Rules are matched to remote station (device) names *and* history names, which determine the corresponding capacity and full policy values to apply upon history creation. If a history does not match any rule, it is created using the same capacity and full policy as in the source history. For an example related to this scenario, see "Example config rules".

Example config rules

As shown in the three-station scenario of Figure 2-4, histories in two JACE stations are exported to the "Supervisor" station. As shown in Figure 2-5, the receiving supervisor station has 2 additional config rules under its History Policies of its NiagaraNetwork.

Figure 2-5 *Example history policies config rules (supervisor station)*



Fox Service	Niagara Fox Service
Tuning Policies	Niagara Tuning Policy Map
History Policies	History Network Ext
Default Rule	Config Rule
Device Pattern	*
History Name Pattern	Audit*
capacity	Unlimited
fullPolicy	Roll
MetersRule	Config Rule
Device Pattern	*
History Name Pattern	Meter*
capacity	Unlimited
fullPolicy	Roll
EastWins	Config Rule
Device Pattern	East*
History Name Pattern	*
capacity	Record Count 100000 records
fullPolicy	Roll

In this example, the highest-priority “Default Rule” matches all (any) stations exporting, with a history name pattern of “`Audit*`”—this matches any “`AuditHistoryLog`”. Capacity is set to unlimited, as all history records are wanted for archive.

The next two rules are applied to histories exported (by any stations), as follows:

- MetersRule — This rule applies to any station, for any history named beginning with “Meter” (“`Meter*`”). Any such history is archived using unlimited capacity, as all records are wanted.
- EastWins — This rule applies only to stations named beginning with “East” (“`East*`”), for any history. Such a history is archived using a capacity of 100,000 and a “roll” full policy.

Following this example (with exported histories in the JACE stations, as shown in Figure 2-4), the histories are created in station “Supervisor” as follows:

- WestWing403
  Histories are exported using following capacity and fullPolicy (from config rule):
  - AuditHistoryLog: unlimited, roll (from Default Rule)
  - LogHistory: 500 records, roll (no rule matched, using source history config, as in JACE)
  - Meter1: unlimited, roll (from “MetersRule”)
  - ZoneTemp1: 500 records, roll (no rule matched, using source history config, as in JACE)
  - Meter2: unlimited, roll (from “Meters Rule”)
  - ZoneTemp2: 500 records, roll (no rule matched, using source history config, as in JACE)
- EastWing403
  Histories are exported using following capacity and fullPolicy (from config rule):
  - AuditHistoryLog: unlimited, roll (from “Default Rule”)
  - LogHistory: 100,000 records, roll (from “EastWins” rule)
  - Meter4: unlimited, roll (from MetersRule)
  - ZoneTemp1: 100,000 records, roll (from “EastWins” rule)
  - Meter3: unlimited, roll (from Meters Rule)
  - ZoneTemp2: 100,000 records, roll (from “EastWins” rule)

Again, note that the rules are processed in priority order. In this example, if the two optional rules were reordered (“EastWins” above “MeterRule”) before the JACE histories were exported, results would differ. “Meter” histories from EastWing403 would have a 100,000 record capacity instead.

Niagara Station Manager notes

The Station Manager is the default view for the Niagara Network. It operates like the Device Manager for most drivers that have online “device discovery” capability. For general information, see “About the Device Manager” on page 1-14.

When using the Station Manager, the following notes apply:

- Station Learn and Discover notes
- Station Add notes

Station Learn and Discover notes

The following sections describe things specific about learn and discover in the NiagaraNetwork:

- Discover (Fox) uses UDP multicasting
- SSL ports (AX-3.7 and later)
- IP address versus hostname

Discover (Fox) uses UDP multicasting

When performing a NiagaraNetwork discover from a Supervisor or an engineering workstation PC, your Windows *firewall* needs exceptions for UDP port(s) as well as TCP port(s), as *UDP multicasting* is used by Fox in discovery. Otherwise (with only a TCP port opened), your discovery *may come up empty*.

Therefore, if using only the "standard" port (1911) for station (Fox) communications, your firewall needs two exceptions:

- TCP port 1911
- UDP port 1911 (needed for a Discover from your NiagaraNetwork)

If a Supervisor host using the "standard" port for platform connections (3011), you might also add an exception for this port too (TCP port 3011)—or for whatever TCP port is used by the platform daemon. Note that UDP multicasting is *not* used in platform communications.

If using "non-standard" ports for station (Fox) connections, note similar firewall exceptions may be needed for those TCP and UDP ports.

SSL ports (AX-3.7 and later) Starting in AX-3.7, secure socket layer (SSL or TLS) connections are supported for Niagara station (Fox) connections, as well as for Niagara platform connections and web browser (HTTP) connections to stations—providing Niagara hosts are properly licensed and configured. Note in this case a station's Fox service can be configured to allow both regular (unsecured) Fox communications as well as secured (*Foxs*) communications, or *only* secured (Foxs) communications.

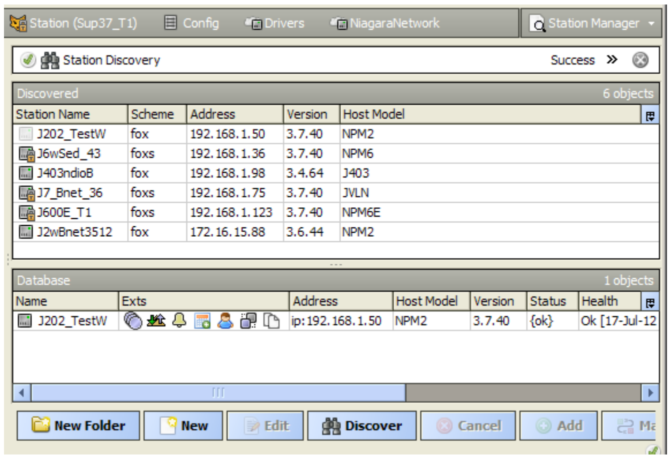
Different ports are used by Foxs than by Fox. By default, Foxs uses TCP port 4911 (where multicasting uses UDP port 4911). This is mentioned because firewall adjustments may be needed to support Niagara station discovery in AX-3.7 and later, depending on job configuration.

For related details, see "About the Fox Service" on page 2-2 and "Discovery notes when stations use secure Fox (Foxs)" on page 2-10. For complete details on SSL in Niagara, refer to the *NiagaraAX SSL Connectivity Guide*.

Discovery notes when stations use secure Fox (Foxs)

Starting in AX-3.7, when performing a NiagaraNetwork discover and one or more remote stations are configured with Foxs (secure Fox), the network's **`Station Manager`** view recognizes this (Figure 2-6).

Figure 2-6 *Station discovery recognizes Foxs (secure Fox) connections*



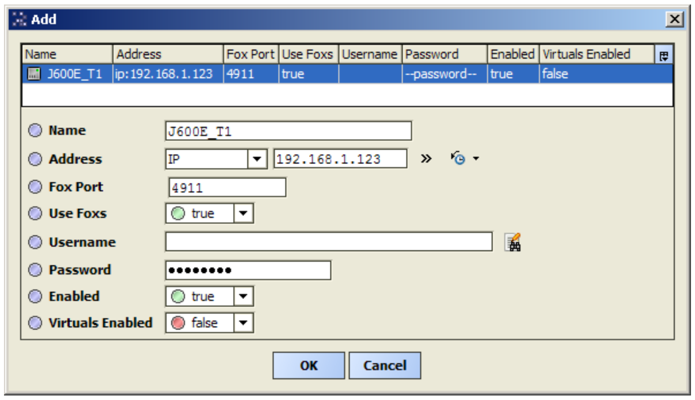
Station Name	Scheme	Address	Version	Host Model
J202_TestW	fox	192.168.1.50	3.7.40	NPM2
J6wSed_43	foxs	192.168.1.36	3.7.40	NPM6
J403ndioB	fox	192.168.1.98	3.4.64	J403
J7_Bnet_36	foxs	192.168.1.75	3.7.40	JVLN
J600E_T1	foxs	192.168.1.123	3.7.40	NPM6E
J2wBnet3512	fox	172.16.15.88	3.6.44	NPM2

Name	Exts	Address	Host Model	Version	Status	Health
J202_TestW		ip:192.168.1.50	NPM2	3.7.40	{ok}	Ok [17-Jul-12

As shown above, such discovered stations show a icon by Station Name and show "foxs" for Scheme. Note that it is possible for a station to be listed *twice*—once with this "foxs" scheme and another with the regular "fox" scheme (and icon without padlock). However, typically you choose to add the "foxs" one.

When you double-click such a station for the **`Add`** dialog (or click to select, then click the **`Add`** button) the **`Add`** dialog will show the port used by that station for Foxs, for example 4911 (Figure 2-7).

Figure 2-7 *Add dialog for NiagaraStation using Foxs (secure Fox)*

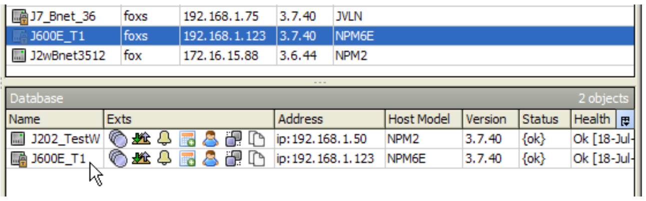


Name	Address	Fox Port	Use Foxs	Username	Password	Enabled	Virtuals Enabled
J600E_T1	ip:192.168.1.123	4911	true		--password--	true	false

Name	J600E_T1
Address	IP 192.168.1.123
Fox Port	4911
Use Foxs	true
Username	
Password	••••••••
Enabled	true
Virtuals Enabled	false

In addition, the "Use Foxs" property is true by default. After entering the station username and password (used for client connection to it), and clicking **OK**, the station is added to the database pane in the **Station Manager** view. The status of the NiagaraStation is ok, providing a "trusted root" or "allowed host exception" is in the local station's platform (for validation of that host's server certificate).

Figure 2-8 *NiagaraStation added for station using Foxs (secure Fox)*



J7_Bnet_36	foxs	192.168.1.75	3.7.40	JVLN
J600E_T1	foxs	192.168.1.123	3.7.40	NPM6E
J2wBnet3512	fox	172.16.15.88	3.6.44	NPM2

Database 2 objects

Name	Exts	Address	Host Model	Version	Status	Health
J202_TestW		ip:192.168.1.50	NPM2	3.7.40	{ok}	Ok [18-Jul-
J600E_T1		ip:192.168.1.123	NPM6E	3.7.40	{ok}	Ok [18-Jul-

Again, such a station shows a icon in the database pane. Note that some jobs may include a *mix* of JACE types, some of which *can support* AX-3.7 or later SSL (JACE-6 series, JACE-7 series, any Win32-based JACE models), while other JACE types *do not* (JACE-2 series, or JACE-4 series, JACE-5 series). Stations running on the latter types lack the ability for a Foxs connection.

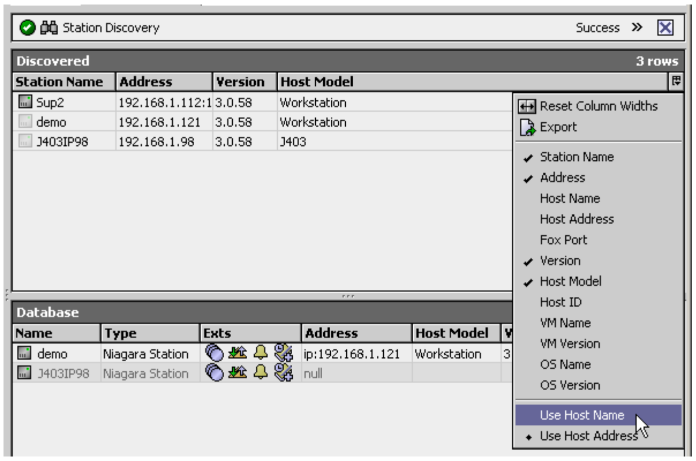
Note: *When configuring any AX-3.7 or later station using SSL (for example a Supervisor station) that requires station communications between a JACE that is not configured for (or does not support) 3.7/3.8 SSL, you must set its FoxService property "Foxs Only" at a* `false` *value. Otherwise, the remote JACE will be unable to make a client connection back to the station. For related details, see "Fox Service properties" on page 2-3.*

For more details on adding a discovered NiagaraStation, see "Station Add notes" on page 2-12.

IP address versus hostname

By default, a station discover uses IP address (vs. host name). Typically, this is the recommended method. However, if Niagara hosts are installed on a network using DNS, you can toggle the discovery method to use host name instead. Do this in the table options menu (click upper right small control) of the Station Manager, as shown in Figure 2-9. You can also select other columns for display, and sort on any column. See "Manager table features" on page 1-21 for more details.

Figure 2-9 *Station Manager table options menu*



Station Discovery — Success

Discovered — 3 rows

Station Name	Address	Version	Host Model
Sup2	192.168.1.112:1	3.0.58	Workstation
demo	192.168.1.121	3.0.58	Workstation
J403IP98	192.168.1.98	3.0.58	J403

Database

Name	Type	Exts	Address	Host Model	V
demo	Niagara Station		ip:192.168.1.121	Workstation	3
J403IP98	Niagara Station		null		

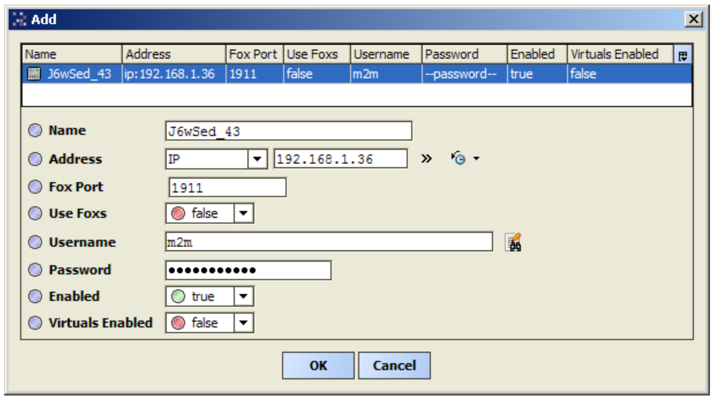
Depending on the chosen discovery method, discovered stations appear listed with either an IP address or a DNS name in the Address column. By default, discovered stations list showing an IP address. If a station uses a Fox port other than 1911 (default), address includes the port number appended, for example: 192.168.1.112.1912.

Station Add notes

When you add a NiagaraStation to the database, the Add dialog includes its Fox port, along with fields for station username and password (required for client connection) as shown in Figure 2-10.

Note: *After you add a station, in the Station Manager's database pane just double-click it to bring up the* **`Edit`** *dialog, which provides access to the same properties in the* **`Add`** *dialog. If needed, you can access all (station) client connection properties from the NiagaraStation component's* property sheet *("Client Connection" slot, along with status properties).*

Figure 2-10 *Add dialog for NiagaraStation*



Add

Name	Address	Fox Port	Use Foxs	Username	Password	Enabled	Virtuals Enabled
J6wSed_43	ip:192.168.1.36	1911	false	m2m	--password--	true	false

Name	J6wSed_43
Address	IP 192.168.1.36
Fox Port	1911
Use Foxs	false
Username	m2m
Password	•••••••••••
Enabled	true
Virtuals Enabled	false

Note: *Starting in AX-3.7, a "Use Foxs" property is present when adding a NiagaraStation, by default set to false—say if manually adding (***`Add`** *button). In this case, set this to true only if the remote host is properly configured for SSL, including the FoxService of the NiagaraNetwork on its station (you must know the port it uses). If using the* **`Discover`** *feature in the* **`Station Manager`**, *any discovered stations using Foxs appears differently, and addition is simplified. See "Discovery notes when stations use secure Fox (Foxs)" on page 2-10. For complete SSL details, refer to the* NiagaraAX SSL Connectivity Guide.

Typically, you enter a user name and password for a specific "service account" user previously made in the remote station. By recommendation, this should be a user that was created especially for *station-to-station* access, typically with admin write privileges, and not otherwise used for normal (login) access of that station. Additionally, this user should *uniquely named* for each project, and have a *strong password*. Refer to the section "Multi-station security notes" in the *User Guide* for related details.

- The "Virtuals Enabled" property determines whether virtual components can be accessed in this station, by any user with permissions on its "Niagara Virtual Gateway". For related details, see "About Niagara virtual components" on page 2-38 and "Security and Niagara virtuals" on page 2-45.
- When working in a Supervisor station configured for Niagara provisioning, additional "platform"

properties are in the **Add** (or **Edit**) dialog, to specify the platform daemon credentials and port the provisioning code should use to connect to the remote JACE platform, to run provisioning jobs. Figure 2-11 shows these properties near the bottom of the Add dialog.

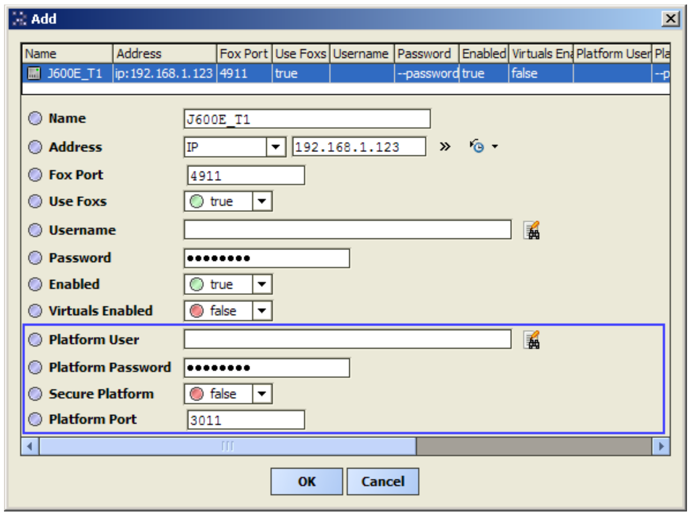


Figure 2-11 *NiagaraStation Add dialog in Supervisor station's NiagaraNetwork (provisioning-configured)*

Add

Name	Address	Fox Port	Use Foxs	Username	Password	Enabled	Virtuals Ena	Platform User	Pla
J600E_T1	ip:192.168.1.123	4911	true		--password	true	false		--p

Name	J600E_T1
Address	IP 192.168.1.123
Fox Port	4911
Use Foxs	true
Username	
Password	••••••••
Enabled	true
Virtuals Enabled	false
Platform User	
Platform Password	••••••••
Secure Platform	false
Platform Port	3011

Briefly, these "platform related" properties in the Add/Edit dialog are described as follows:

- Platform User — User account for the station's platform, the same used as when making a platform connection to it using Workbench.
- Platform Password — Password for this user account.
- Secure Platform — (new starting in AX-3.7) A Boolean that is false by default. Set to true only if the JACE is currently configured to support a secure (SSL) platform connection.
- Platform Port — Port this JACE's platform daemon monitors for a platform connection, with the default value of 3011 ("standard" port for a "regular", or unsecure platform connection). If using a non-standard platform port, or a secure (SSL) platform connection, you edit this to match that port number. Note the "standard" port for platform SSL is 5011, but this may be configured differently in the JACE.

For more details, refer to the document *NiagaraAX Provisioning Guide for Niagara Networks.*

Note: *Adding a NiagaraStation automatically creates a reciprocal NiagaraStation in the remote station. See the section "Reciprocal NiagaraStation component".*

Reciprocal NiagaraStation component

When you add a station under the Niagara Network, that remote station *automatically adds* a "reciprocal" NiagaraStation component under its *own* Niagara Network, representing the *original* station. However, its "Enabled" property is *false*, with a "disabled" status (looks grayed out as in Figure 2-12).

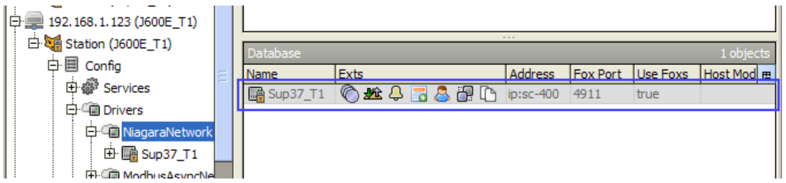


Figure 2-12 *Example reciprocal NiagaraStation (representing Supervisor) created in JACE station*

Database 1 objects

Name	Exts	Address	Fox Port	Use Foxs	Host Mod
Sup37_T1		ip:sc-400	4911	true	

This provides a visual clue for you to edit its **Client Connection** properties Username and Password to valid credentials for an appropriate user in the reciprocal station, and also to set the Enabled property of the NiagaraStation from false to true (to allow operation).

Note: *As shown in Figure 2-12, sometimes a reciprocal NiagaraStation may get added using "hostname" for Address, rather than its preferred IP address. It is recommended that you edit the NiagaraStation's Address back to IP format, from the NiagaraStation's property sheet. Otherwise, the station may remain "down" after you enable it.*

NiagaraStation component notes

This section explains items unique to NiagaraStation components versus other device components. Note that NiagaraStations each contain the standard set of device extensions (see "Types of device extensions" on page 1-26).

- About station status properties
- About client connection properties
- About server connection properties
- About station "provisioning" extensions

Note: *The following additional NiagaraStation items are found in later NiagaraAX releases*

- NiagaraStations also have a "`Users`" device extension. See "About the Users extension" on page 2-15.
- NiagaraStations also have a "`Virtuals`" (gateway) slot, and an associated "Virtuals Enabled" property. See "About Niagara virtual components" for an overview, as well as sections "About the Niagara Virtual Device Ext" on page 2-40 and "Security and Niagara virtuals" on page 2-45.

About station status properties

Station status properties are typical of most device status properties. See "Device status properties" on page 1-22. If a NiagaraStation has a down status, typically the "Health" container's "Last Fail Cause" property value provides a clue why.

About client connection properties

Note: *In 2013 "update" releases of NiagaraAX (including AX-3.7u1), the password of a NiagaraStation's Client Connection is encrypted in a "non-portable" manner, as any other stored client passwords (for example, password in the OutgoingAccount under the EmailService). This means that you cannot simply save the station database (`config.bog`), and use it on another host with these client passwords still functional.*

However, note this does not *apply to a AX-3.8 station, where client passwords in a copied station (`config.bog` file)* are *portable. For details, refer to the* NiagaraAX 2013 Security Updates *document.*

Client connection properties specifiy the Fox port used for station and Workbench connections, as well as the user name and password used for station-to-station connections.

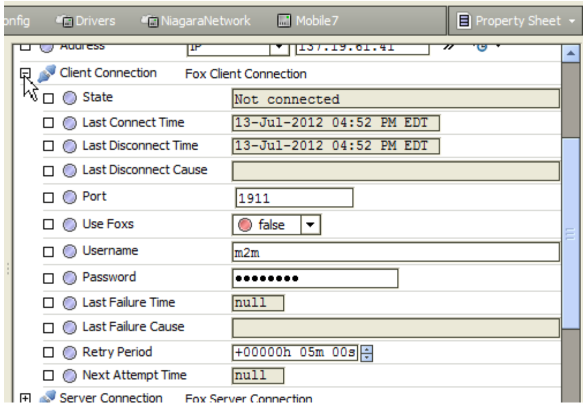


Figure 2-13 *Client connection container expanded in NiagaraStation property sheet*

Address	IP 137.19.81.41
Client Connection	Fox Client Connection
State	Not connected
Last Connect Time	13-Jul-2012 04:52 PM EDT
Last Disconnect Time	13-Jul-2012 04:52 PM EDT
Last Disconnect Cause	
Port	1911
Use Foxs	false
Username	m2m
Password	●●●●●●●●
Last Failure Time	null
Last Failure Cause	
Retry Period	+00000h 05m 00s
Next Attempt Time	null
Server Connection	Fox Server Connection

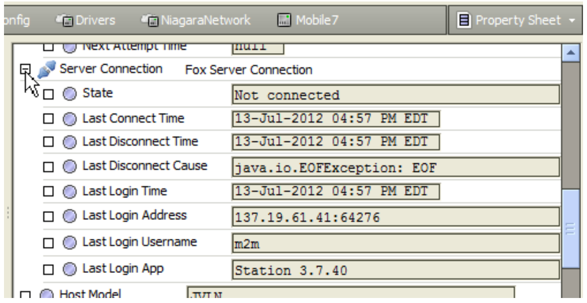
Figure 2-13 shows the Client Connection container expanded in a NiagaraStation's property sheet. Many properties are status types, that contain either real-time or historical information.

Right-click actions on the Client Connection container let you **`Manual Connect`** or **`Manual Disconnect`** to this remote station.

About server connection properties

Server connection properties are status properties that provide either real-time or historical information. Figure 2-14 shows the Server Connection container expanded in a NiagaraStation's property sheet.

Figure 2-14 *Server connection container expanded in NiagaraStation property sheet*



onfig | Drivers | NiagaraNetwork | Mobile7 | Property Sheet

Next Attempt Time	null
Server Connection	Fox Server Connection
State	Not connected
Last Connect Time	13-Jul-2012 04:57 PM EDT
Last Disconnect Time	13-Jul-2012 04:57 PM EDT
Last Disconnect Cause	java.io.EOFException: EOF
Last Login Time	13-Jul-2012 04:57 PM EDT
Last Login Address	137.19.61.41:64276
Last Login Username	m2m
Last Login App	Station 3.7.40
Host Model	JVLN

A right-click action on the Server Connection provides a **`Force Disconnect`** command.

About station “provisioning” extensions

Any Supervisor station that is configured for “provisioning” has NiagaraStation components that each contain *provisioning* device extensions. For more details, see “Types of provisioning extensions” in the document *Provisioning for Niagara Networks*.

Note that provisioning extensions do *not* appear in the Station Manager, but do appear in the Nav tree.

About the Users extension

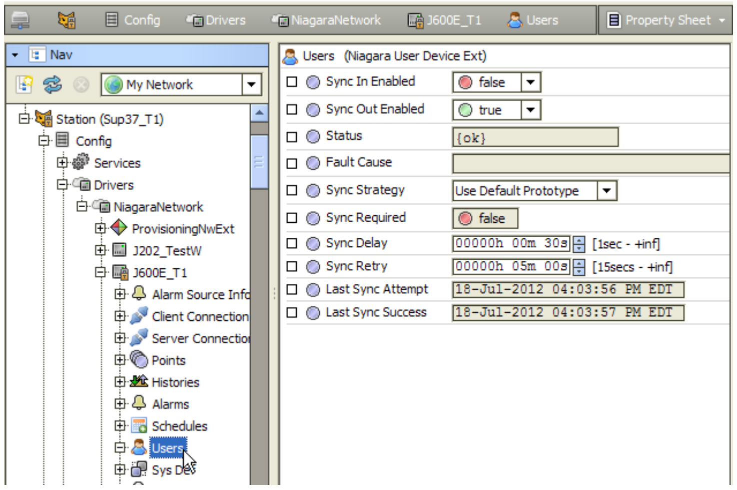
Any NiagaraStation has a “**`Users`**” device extension. This device extension is currently unique to the Niagara driver. It can be used in multi-station jobs where “centralized management” of station users is needed. For an overview of this user security feature, refer to the section “Network users” in the *User Guide*.

Note there is no special view (apart from property sheet) on the Users device extension, nor is it a container for other components. Instead, its properties specify the “sync” operation for “network users” in the proxied station, relative to the current station.

For more details, see the following subsections:

- About Users extension properties
- Example Sync In and Out values
- About Users sync strategy
- Users device extension configuration notes

Figure 2-15 *Users device extension of NiagaraStation*



Users (Niagara User Device Ext)

Sync In Enabled	false
Sync Out Enabled	true
Status	{ok}
Fault Cause	
Sync Strategy	Use Default Prototype
Sync Required	false
Sync Delay	00000h 00m 30s [1sec - +inf]
Sync Retry	00000h 05m 00s [15secs - +inf]
Last Sync Attempt	18-Jul-2012 04:03:56 PM EDT
Last Sync Success	18-Jul-2012 04:03:57 PM EDT

About Users extension properties

Properties include the following:

- **Sync In Enabled**
  Default is false. Set to true if the proxied NiagaraStation “sends” users to current station. Otherwise leave at false.

- **Sync Out Enabled**
  Default is false. Set to true if the proxied NiagaraStation "receives" users from current station. Otherwise leave at false.
  Note: *You cannot set both sync properties to true, or else a fault will occur. Also, note that if setting one of these to true, the "reciprocal" property must also be set to "true" in the remote (proxied) station, in the Users extension of the NiagaraStation that represents the current station. See "Example Sync In and Out values" on page 2-16.*
- **Status**
  Either (ok} or {fault} as status of Users device extension.
- **Fault Cause**
  If Users extension is in fault, provides the reason. Note you can see a fault when first enabling a Sync In or Sync Out property, until you enable the reciprocal Sync In or Sync Out property in the remote station.
- **Sync Strategy**
  As one of the following:
  - Prototype Required - (Default) A network user is not added or modified on this proxied station unless that remote station has a matching (identically-named) User Prototype, as referenced in the source User's "Prototype Name" property.
  - Use Default Prototype - A network user always added or modified. If the remote station has a User Prototype named the same as the User's "Prototype Name", that user prototype is used, otherwise the "Default Prototype" in the receiving station is used.
- **Sync Required**
  A read-only boolean that indicates if pending user changes require a sync to become effective. Ordinarily false unless user changes have occurred, and the sync delay time has not expired.
- **Sync Delay**
  Default is 30 seconds. Applies within a "sending user" station only. Specifies a configurable delay time that counts down following a change to a network User. Resets to full delay time upon each successive change. Following the last user change, if the delay time is met, network user changes (sent to the proxied station) are synchronized. This can be set uniquely among proxied stations if needed.
- **Sync Retry**
  Default is 5 minutes. Applies within a "sending user" station only. Specifies a configurable repeat time for previously unsuccessful user synchronization attempts (sent to the proxied station). Periodically, user synchronization will be attempted at this frequency until a successful sync is recorded.
- **Last Sync Attempt**
  Read-only timestamp of when user synchronization to/from this station was last attempted.
- **Last Sync Success**
  Read-only timestamp of when user synchronization to/from this station was evaluated as successful.

Example Sync In and Out values

Typically, with a Supervisor as the sole "user sending" station, and multiple JACE stations where each is a "user receiving" station, Sync In and Sync Out property configuration of the Users device extension of NiagaraStations is done in the different station databases as follows:

WebSup (Supervisor station):

```
Drivers
└──NiagaraNetwork
   ├──JstationA
   │  └──Users
   │     ├──Sync In Enabled = false
   │     └──Sync Out Enabled = true
   ├──JstationB
   │  └──Users
   │     ├──Sync In Enabled = false
   │     └──Sync Out Enabled = true
   └──JstationC
      └──Users
         ├──Sync In Enabled = false
         └──Sync Out Enabled = true
```

JstationA (JACE station, typical of others):

```
Drivers
└─NiagaraNetwork
    ├─WebSup
    │  └─Users
    │     ├─Sync In Enabled = true
    │     └─Sync Out Enabled = false
    ├─JstationB
    │  └─Users
    │     ├─Sync In Enabled = false
    │     └─Sync Out Enabled = false
    └─JstationC
       └─Users
          ├─Sync In Enabled = false
          └─Sync Out Enabled = false
```

Note it also possible to have a “multi-tier” network user sync setup, for example where there is a single “master” Supervisor host and several “subordinate” Supervisor hosts, each with multiple subordinate JACE stations. In the NiagaraNetwork configuration of each subordinate Supervisor station, it would be enabled to “Sync In” to the single NiagaraStation that proxies the master Supervisor. In other NiagaraStations that proxy remote JACEs, their Users extension is enabled to “Sync Out”. In this regard, each subordinate Supervisor station could be considered both “user receiving” and “user sending”.

MasterSup (“Master Supervisor” station):

```
Drivers
└─NiagaraNetwork
    ├─subWebSupA
    │  └─Users
    │     ├─Sync In Enabled = false
    │     └─Sync Out Enabled = true
    ├─subWebSupB
    │  └─Users
    │     ├─Sync In Enabled = false
    │     └─Sync Out Enabled = true
    └─JstationA
       └─Users
          ├─Sync In Enabled = false
          └─Sync Out Enabled = true
```

subWebSupA (“Subordinate Supervisor” station A):

```
Drivers
└─NiagaraNetwork
    ├─MasterSup
    │  └─Users
    │     ├─Sync In Enabled = true
    │     └─Sync Out Enabled = false
    ├─subWebSupB
    │  └─Users
    │     ├─Sync In Enabled = false
    │     └─Sync Out Enabled = false
    └─JstationB
       └─Users
          ├─Sync In Enabled = false
          └─Sync Out Enabled = true
```

subWebSupB ("Subordinate Supervisor" station B):

```
Drivers
└─NiagaraNetwork
    ├─MasterSup
    │  └─Users
    │     ├─Sync In Enabled = true
    │     └─Sync Out Enabled = false
    ├─subWebSupA
    │  └─Users
    │     ├─Sync In Enabled = false
    │     └─Sync Out Enabled = false
    └─JstationC
       └─Users
          ├─Sync In Enabled = false
          └─Sync Out Enabled = true
```

About Users sync strategy

By default, the “sync strategy” for network users (in the Users device extension of a NiagaraStation) is configured for the “Prototype Required” method. This can be useful to reduce the number of distributed network users, whenever a source User is configured to have a non-default “Prototype Name”. Only user-receiving stations that have a matching-named User Prototype will replicate that user, using the 3 "local override" properties of that local prototype. In addition, this method allows greater variations for defining different schemes for permissions, navigation, and Web profile types among groups of users.

However, you can simplify things by configuring sync strategy for "Use Default Prototype". This will result in more replicated network users created, as a user-receiving station will simply use its Default Prototype for a received User (if it does not have a matching-named User Prototype).

See the section "Sync process description" on page 2-19 for details on how the user synchronization feature works. For related details on the "UserService side" of network users, refer to the section "About User prototypes" in the *User Guide*.

Users device extension configuration notes

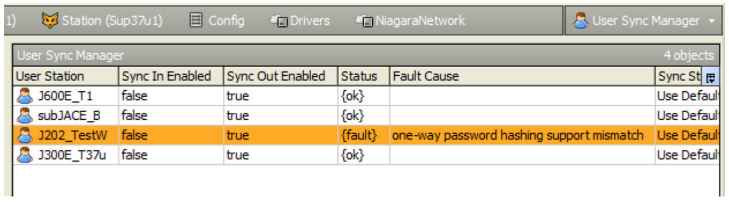
When setting up the NiagaraNetwork portion of user synchronization between stations in Workbench (in each Users device extension), first open up all stations in Workbench so you can expand the NiagaraNetwork in each station—both Supervisor and JACEs stations. Remember that Sync In and Sync Out is set up in an *opposite* fashion on reciprocal stations.

The easiest way to do this, particularly on the Supervisor station, is using the **`User Sync Manager`** view. There, you typically select all (JACE) stations in this view to have a "Sync Out Enabled" of true. See the next section "About the User Sync Manager" for more details.

Update mismatch with Users device extension fault

When upgrading a system to a 2013 security "update" level (typically AX-3.7u1 or later), stations running on JACEs not yet upgraded to an update release level will not be able to sync network users from an upgraded Supervisor (or other upgraded station). Instead, the Users device extensions of the NiagaraStations will be in fault, with a fault cause of "`one-way password hashing support mismatch`".

Figure 2-16 *Upgrade mismatch results in Users device extension fault*



User Sync Manager — 4 objects

User Station	Sync In Enabled	Sync Out Enabled	Status	Fault Cause	Sync St
J600E_T1	false	true	{ok}		Use Defaul
subJACE_B	false	true	{ok}		Use Defaul
J202_TestW	false	true	{fault}	one-way password hashing support mismatch	Use Defaul
J300E_T37u	false	true	{ok}		Use Defaul

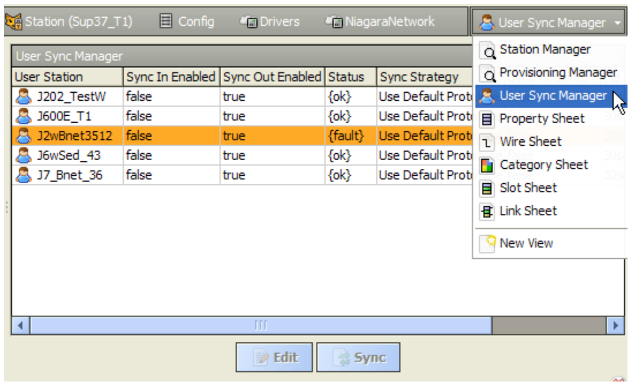
However, once a JACE is upgraded to a 2013 update level (typically AX-3.7u1 or later), network user operation with an upgraded Supervisor station will continue as normal, including sync operation.

About the User Sync Manager

Any station's NiagaraNetwork has an available **`User Sync Manager`** view. Select it by right-clicking the NiagaraNetwork for its *Views* menu, or by using the view selector from any NiagaraNetwork view, as shown in Figure 2-17.

This view reflects the "NiagaraNetwork" side setup for enabling "network users." For an overview of this feature, refer to the section "Network users" in the *User Guide*.

Figure 2-17 *User Sync Manager is available view on NiagaraNetwork*



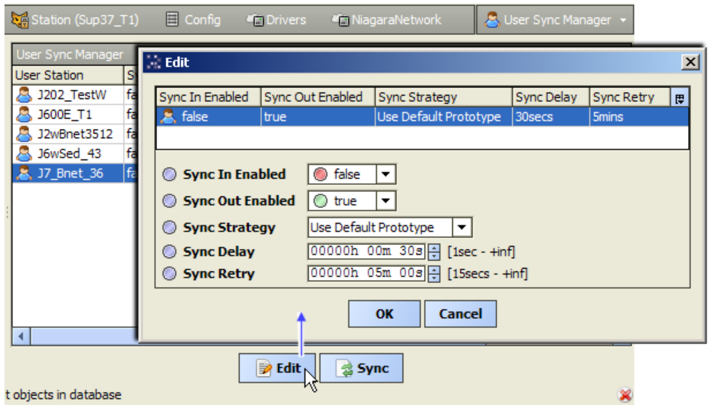
User Sync Manager

User Station	Sync In Enabled	Sync Out Enabled	Status	Sync Strategy
J202_TestW	false	true	{ok}	Use Default Prot
J600E_T1	false	true	{ok}	Use Default Prot
J2wBnet3512	false	true	{fault}	Use Default Prot
J6wSed_43	false	true	{ok}	Use Default Prot
J7_Bnet_36	false	true	{ok}	Use Default Prot

For each NiagaraStation in the network, the `User Sync Manager` shows the current status and configuration property values of its Users device extension. The example above, with its configuration of "Sync In Enabled" and "Sync Out Enabled" properties, is representative of a NiagaraNetwork in a *Supervisor* station, where all child stations are subordinate (typically JACE) stations.

From this view you can select one or more rows (stations) to edit properties, as shown in Figure 2-18 below, or manually request a sync of selected stations.

Figure 2-18 *Edit dialog for one or more selected station's Users extension*



Sync In Enabled	Sync Out Enabled	Sync Strategy	Sync Delay	Sync Retry
false	true	Use Default Prototype	30secs	5mins

Sync In Enabled	false	
Sync Out Enabled	true	
Sync Strategy	Use Default Prototype	
Sync Delay	00000h 00m 30s	[1sec - +inf]
Sync Retry	00000h 05m 00s	[15secs - +inf]

Sync process description

User-sending station (Supervisor) side

1. All Users device extensions with Sync Out Enabled=true receive notification that a user event has occurred. This results in setting Sync Required=true in each Users device extension.
2. A synchronization is scheduled for the remote station. The Sync Delay property determines how long to wait between a user event and a synchronization. Each event resets the delay. That allows multiple changes to be made without necessarily requiring multiple synchronization operations.
3. After the Sync Delay has passed, the Supervisor initiates the synchronization process with the subordinate station.
4. When complete, Sync Required=false and sync timestamps are updated.

User-receiving station (e.g. JACE) side

1. Subordinate checks Sync In Enabled of its proxy for the Supervisor station. If false, the synchronization process is aborted.
2. Subordinate coordinates with Supervisor to determine which users are out of sync.
3. Subordinate receives list of out-of-sync user records from Supervisor.
4. For each user:
   - Find a matching prototype among User Prototypes. A prototype is either matched by name or if the Sync Strategy is "Use Default Prototype", the default prototype is used. If no matching prototype, ignore user.
   - On the incoming user, overwrite the "local override properties" (properties: Permissions, Nav File, Web Profile) from the prototype.
   - Add the new user or update an existing user.
5. When complete, Sync Required=false and sync timestamps are updated.

Note:*For related details, refer to the section "About User prototypes" in the* User Guide*.*

Niagara Discover enhancements

The Discover feature in manager views for Niagara proxy points, histories, and schedules, was enhanced starting in AX-3.5, such that discovery is essentially performed by the *station* you have opened in Workbench. Previously, such a discovery initiated a Fox connection directly between Workbench and the target remote station—you may have noticed another Fox station (connection) node in the Nav tree.

Typically, you perform operations like this from an opened *Supervisor* station.

This discovery enhancement solves the following possible issues:

- Firewalls allow station-to-station connections, but block Workbench-to-remote station connections.

- Branding restrictions, for example where "appliances" allow a station-to-station connection, but prevent direct Workbench-to-remote station connection.

Note: *In addition, this change lets you perform discovery of Niagara proxy points, histories, and schedules using* browser access, *that is, "Web Workbench". Prior to AX-3.5, an error message resulted if this was attempted.*

Note the following about these discovery enhancements:

- No related changes are visible in the various manager views, such as **`Niagara Point Manager`**, **`Niagara History Import Manager`**, **`Niagara Schedule Import Manager`**, and so on.
- Access to discoveries in the endpoint (target) station requires station login credentials. If that station has an account with the same credentials as the account you used to open the current (Supervisor) station, the discovery process starts immediately, without login.
  If those login credentials fail, the Discover presents a login dialog. Again, you enter the login credentials for that *endpoint* (typically JACE) station.

Niagara Point Manager notes

The Niagara Point Manager is the default view for the **`Points`** device extension (NiagaraPointDeviceExt) under a NiagaraStation. It operates similar to the Point Manager for most drivers that have online "proxy point discovery" capability. For general information, see "About the Point Manager" on page 1-37.

When using the Niagara Point Manager, the following notes apply:

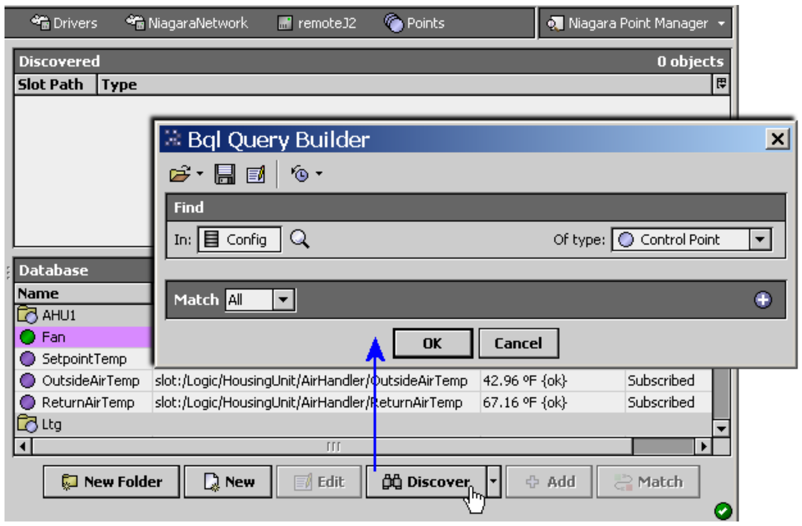
- Niagara point Discover notes
- About the Bql Query Builder

Niagara point Discover notes

Note: *Niagara proxy point discovery was enhanced starting in AX-3.5 to permit operations that were previously blocked or unavailable. For details, see "Niagara Discover enhancements" on page 2-19.*

Niagara point discovery launches the **`Bql Query Builder`**, where you specify the root level of component hierarchy in the remote station to begin the discover.

Figure 2-19 *Discover in Niagara Point Manger uses Bql Query Builder*

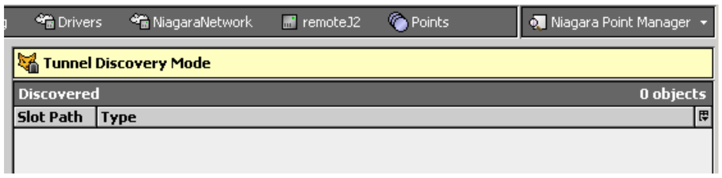


Component type filters are available in that dialog too—for details, see "About the Bql Query Builder" on page 2-21.

Point Discover notes if Fox tunneling (AX-3.4 system)

Starting in AX-3.4, the **`Niagara Point Manager`** was enhanced with an option for point discovery to a tunnel-accessible only station, using a "Tunnel Discovery Mode". Special indication of this mode being active is provided by a banner at the top of the Point Manager, as shown in Figure 2-20.

Figure 2-20 *Tunnel Discovery Mode available, showing indication banner*



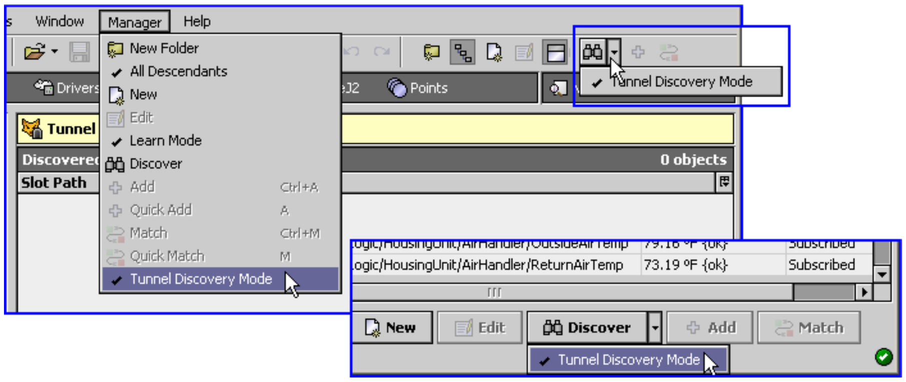
Note: *This feature is not required when connecting to an AX-3.5 or later station and doing point discoveries (and other discoveries). For related details, see "Niagara Discover enhancements" on page 2-19.*

However, if working with an AX-3.4 station (even if using a more recent Workbench), you may need to use this feature for Point discovery for stations accessible only via Fox tunneling. Otherwise, Workbench needs a direct Fox connection for point discovery, that is, be on same LAN.

In this case, you can toggle into this "Tunnel Discovery Mode" to perform a Niagara point discovery.

You enable/disable this feature using a pull-down toggle option on the bottom **`Discover`** button, also available on the Manager menu and toolbar.

Figure 2-21 *Toggle Tunnel Discovery Mode using menu, toolbar, or Discover button pull-down*



For related details, see "About Fox Tunneling" in the engineering notes document *Fox Tunneling and HTTP Tunneling*.

About the Bql Query Builder

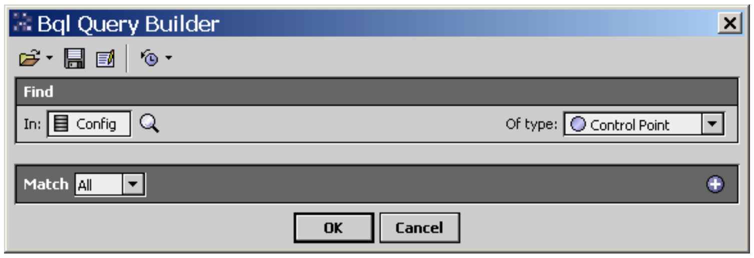
The **`Bql Query Builder`** is the initial dialog you use in the **`Niagara Point Manager`** when you "Discover" proxy points in a station under the Niagara Network. You also use it in discovers in other managers, for example, the Bacnet Export Manager.

This dialog provides extensive Find and Match filters, plus other controls, which are briefly discussed in the following sections:

- About Bql Query Find filters
- About Bql Query Match filters
- About Bql Query Save, Load, and Edit

Note that the default point Discover for a NiagaraStation produces a Bql Query Builder dialog with a *Find* of (all) "Config," type "Control Point," and *Match* of "All," as shown in Figure 2-22. While this may be usable, typically you narrow this range, and run multiple point discovers, as needed.

Figure 2-22 *Default Bql Query Builder from NiagaraStation point Discover*

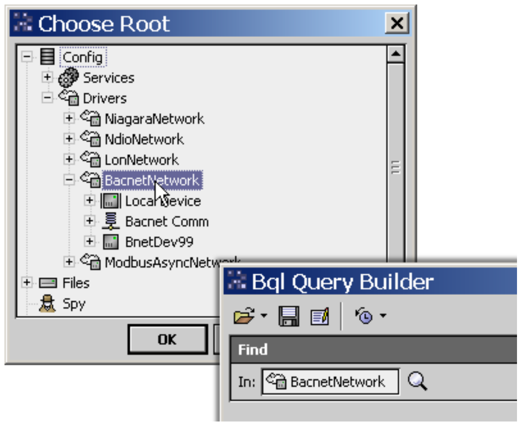


About Bql Query Find filters

By default, the Find filter preselects all control points in the target NiagaraStation (Config). Often, you may wish to narrow this range. For example, you may wish to find Boolean proxy points only under a specific driver network (during this particular Discover sequence).

To do this, simply click the Find icon (magnifying glass) which produces the Choose Root dialog, as shown in Figure 2-23.

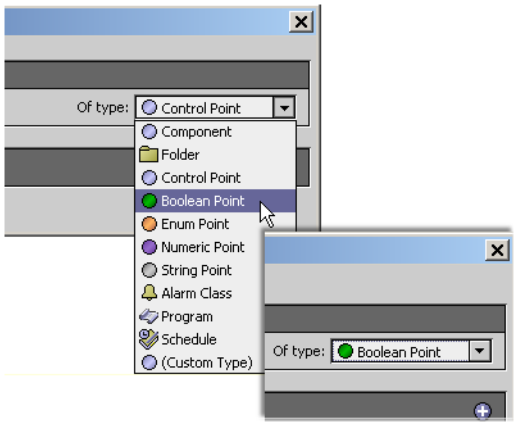
Figure 2-23 *Choose Root dialog*



As needed, expand Config to select the root component of interest and click **OK** (or simply double-click the component of interest). The proxy point Find is now limited to that area of the station.

To change the *type* of component, click the "Of type" drop-down and make a selection (Figure 2-24).

Figure 2-24 *Select component type*



The proxy point Find is now limited to that class of component.

Note: *A basic understanding of the NiagaraAX component class structure is helpful when making type selections. For example, selection of Boolean Point (as shown in Figure 2-24) includes all components that "subclass" from the simple* `BooleanPoint`*. Included are all* `BooleanWritables`*, as well as many kitControl components (Logic components, as one example in this case).*

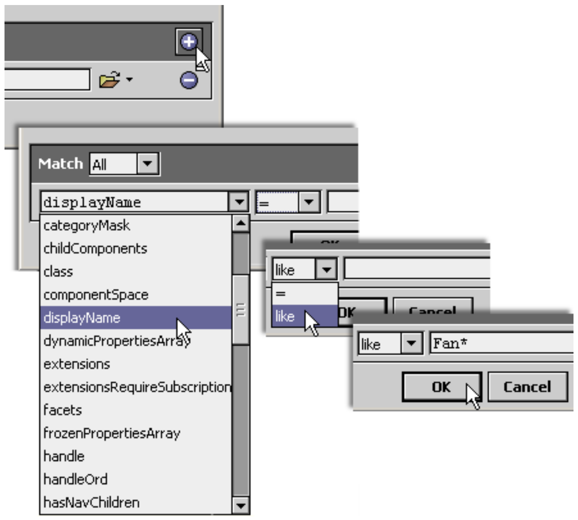
If you select type "`Component`*," you have a "full-width find." This means that all components are included—this includes everything listed in the "Of type" drop-down, plus extensions and many other items (including kitControl objects not subclassed from Control Points, for example, "NumericConst," "DegreeDays," and "LeadLagRuntime," as a few examples).*

About Bql Query Match filters

Use the Match filter when you wish to further limit proxy point candidates. For example, you may wish to filter on object names (this translates to *displayNames*).

To see the match dialog options, click the plus ("+") icon in the far right corner of the Bql Query Builder dialog, and use Match entry fields as needed (Figure 2-25).

Figure 2-25 Expand Match to see fields, use entry fields as needed



In the Figure 2-25 example above, the Match filter is set to: `displayName`, `like`, "`Fan*`". This point discover returns components (within the Find parameters) that are named beginning with "Fan".

This match would include all components named "Fan", "FanAhu1", "Fan21", "Fantastic", and so on. However, components named "FAN" or "FAN21" would *not* be included (case-sensitivity), nor would components named "AhuFan" or "BFan" be included—no leading wildcard ("*") was used.

Note: *You can click the Match plus ("+") icon multiple times to add multiple match lines, and configure each match line differently, as needed. Click the minus "–" icon to remove a match line.*

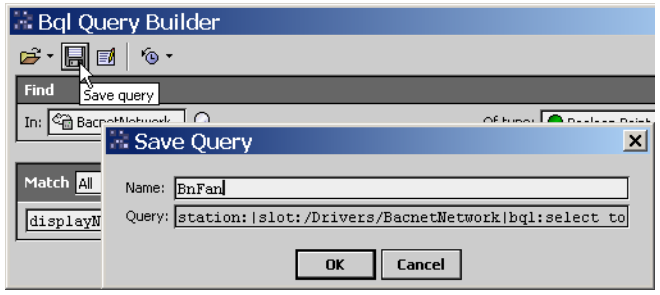
If you have multiple match lines, note the drop-down selection beside **`Match`** ("`All`", "`Any`") becomes *important*, and works as follows:

- `All` — Works as "AND" logic, where a match must occur as specified on *every* match line.
- `Any` — Works as "OR" logic, where a match must occur as specified on *any* match line.

About Bql Query Save, Load, and Edit

Save the setup of any Bql query by simply clicking the **`Save query`** icon (diskette). This produces a **`Save Query`** dialog in which you give the query a unique name (Figure 2-26).

Figure 2-26 Save and name Bql Query



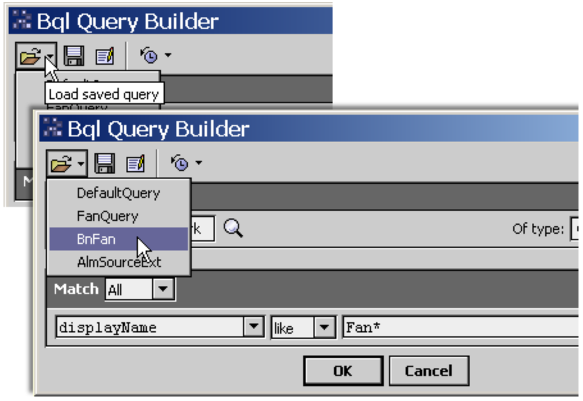
Name: BnFan
Query: station:|slot:/Drivers/BacnetNetwork|bql:select to

Saving a query allows you to recall it later to either use directly, or to modify further as a "starting point" for another query. You can save as many Bql queries as you need. You can also edit a saved query (meaning rename it or reorder it in your list of saved queries).

Note: *Saved queries are unique to your Workbench instance, and not to any particular station.*

To recall a saved query, click the **`Load saved query`** icon (folder), and make a selection, as shown in Figure 2-27.

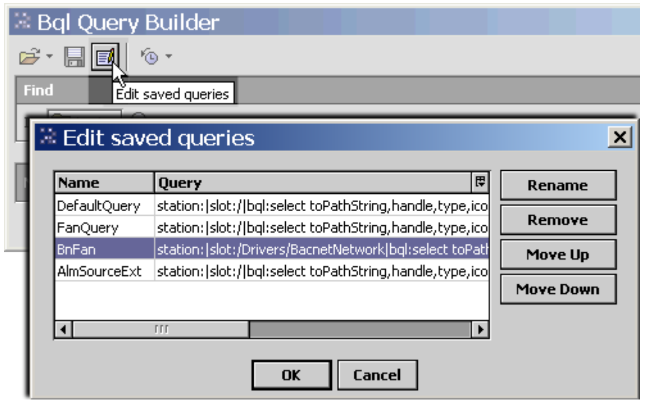
Figure 2-27 Load saved Bql Query



This loads that query into the Bql Query Builder dialog, where Find and Match entries automatically change to reflect how that query was constructed.

To edit saved queries, click the **`Edit saved query`** icon (note pad). This produces a **`Save Query`** dialog in which you can rename and/or reorder the queries in the list (Figure 2-28).

Figure 2-28 Edit saved Bql Queries



Name	Query
DefaultQuery	station:|slot:/|bql:select toPathString,handle,type,ico
FanQuery	station:|slot:/|bql:select toPathString,handle,type,ico
BnFan	station:|slot:/Drivers/BacnetNetwork|bql:select toPat
AlmSourceExt	station:|slot:/|bql:select toPathString,handle,type,ico

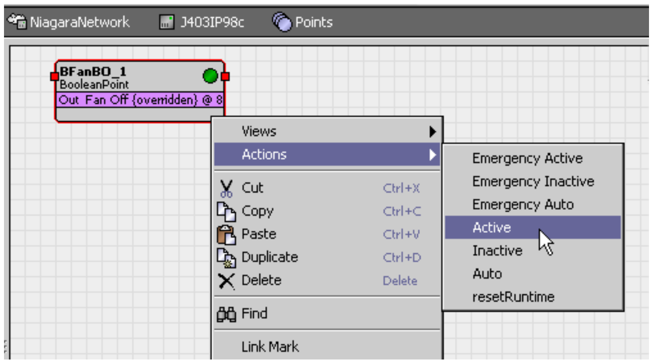
Note: *If you are interested in the Baja Query Language (BQL), you can learn about queries within the Edit dialog. Make the dialog window wider, and study the syntax of a saved queries. For detailed information about queries, see "BQL" in the* Niagara Developer Guide.

Niagara proxy point notes

Note: *An alternative to Niagara proxy points exists for a Supervisor—see "About Niagara virtual components" on page 2-38.*

Niagara proxy points are always read-only types (BooleanPoint, EnumPoint, NumericPoint, and StringPoint). No other types are available. However, if the source component is a *writable* point type, any actions (commands) for that component are available in the Niagara proxy point, as shown in Figure 2-29.

Figure 2-29 Niagara proxy point for writable point includes actions



Other concepts about Niagara proxy points are explained in the following sections:

- Best practices for Niagara proxy points
- Proxy of a proxy, other candidates
- Link control and Niagara proxy points

Best practices for Niagara proxy points

- Are they even needed for Px views?
- Avoid point extensions under Niagara proxy points
- Avoid links to proxy point actions
- When a Niagara proxy point is required

Are they even needed for Px views?

The typical "large scale" usage of Niagara proxy points has historically been in a Supervisor station, such that real-time values could be modeled and thus displayed centrally in Px views on the Supervisor. Prior to AX-3.4, and especially AX-3.5, it was not uncommon for a such a station to have several hundred, or even thousands, of Niagara proxy points—with data originating within various remote JACE stations.

However, with the introduction of "export tags" in AX-3.5, as well as the enhanced, writable "Niagara virtual components", such usage for proxy points in a Supervisor should diminish, for these reasons:

- If a Supervisor needs to serve up PxPages that already exist in a JACE station, simple use of "Px-ViewTag" export tags in that JACE station can *automatically* replicate all needed components on the Supervisor—including all necessary Niagara *virtual* points. Niagara proxy points are not needed. For further details, refer to the document *NiagaraAX Export tags*.
- If engineering PxPages that exist only on the Supervisor, and real-time values and access to right-click actions are needed from subordinate JACE stations (that would previously require Niagara proxy points), consider using *Niagara virtuals* instead. In some cases, virtual components offer techniques previously hard to do when using proxy points, such as providing a user edit access in a graphic of items like alarm limits. See "About Niagara virtual components" on page 2-38.

Avoid point extensions under Niagara proxy points

Although there are no "rules" against it, adding history extensions or alarm extensions to Niagara proxy points may be considered *unwise*, as even momentary communications issues between a JACE station and its Supervisor can result in loss of (otherwise recorded) history records or alarm conditions.

Therefore, the following recommendations apply to configuring Niagara proxy points:

- Instead of adding a history extension to a Niagara proxy point in the Supervisor station, add it instead to the source point in the remote station. Then, either *import* such histories into the Supervisor (or, from the JACE station side, export them to the Supervisor).
  For related details, see "Niagara History Import Manager" on page 2-30, "Niagara History Export Manager" on page 2-33, and "About History Policies" on page 2-7.
- Instead of adding an alarm extension to a Niagara proxy point in the Supervisor station, add it instead to the source point in the remote station. Then, configure alarm routing from the JACE station into the Supervisor. For related details, see "NiagaraStation Alarms notes" on page 2-27.

Note that routines for interstation history archiving (import) and alarm routing routines have integral "retry" mechanisms, such that temporary loss of station communications typically does *not* result in loss of history data or alarm events. Instead, that data safely resides on the JACE until communications can be re-established.

Avoid links to proxy point actions

Although currently the wiresheet and Link Editor allows you to link to *action* slots on Niagara proxy points, this has been found to result in issues, as *such links can be lost*, typically upon a station restart. Unlike property slots, actions can be dynamically refreshed (rebuilt). If such an action was a link target, that link record is gone. An example scenario is in a Supervisor station, where a "master" NumericWritable component's set action has been linked to the set action of multiple Niagara proxy points (where each in turn represents a setpoint of some data item proxied in a driver's network).

Future NiagaraAX releases may prevent linking to proxy point actions in the Link Editor and wiresheet. Until then, *avoid links to actions* of proxy points.

Equivalent link control may be possible by creating Niagara proxy points in each JACE station (of source items in the Supervisor station), and then linking properties of those proxy points to other driver's proxy points in that JACE station. For related details, see "Link control and Niagara proxy points" on page 2-26.

When a Niagara proxy point is required

A Niagara proxy point *is* required whenever you need to use its data value in local station control logic, 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 Niagara proxy points. Niagara virtual components cannot be used in this way, because they do not "persist"—they exist only during active subscriptions (typically from users accessing PxPages).

Sometimes, this type of Niagara proxy point usage may be needed in JACE stations that need to directly share (and process) selected data items for control reasons. For a related topic, see "Link control and Niagara proxy points" on page 2-26.

Proxy of a proxy, other candidates

Often, the (remote) source component of a Niagara proxy point is itself a proxy point, typically under the Points extension of device in a field bus driver network (Lonworks, Bacnet, and so on). Or, if the remote station is in a JACE with its own I/O, the source component may be an Ndio or Nrio 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 of a NumericTotalizerExt. In other cases, the (remote) source component may be a kitControl type, such as one of the Math or Logic types, or one of the other more complicated types (for example, a LoopPoint or DegreeDays object).

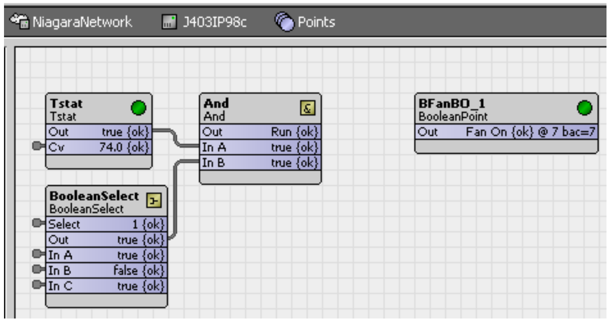
Regardless of the source component, you model all remote Niagara data with proxy points by selecting from the "atomic" model of the four read-only point types.

Link control and Niagara proxy points

Because Niagara proxy points have no inputs, you *cannot link into them* from local station logic (even if the remote source component is a writable type). If you need this "inter-station link control," you must make *another* Niagara proxy point (in the *remote station*) that proxies whatever local source component you need for the link.

Consider a Niagara proxy point for a BooleanWritable that controls a fan. You wish to provide link control from a local And object, controlling at priority level 7. Figure 2-30 shows these objects.

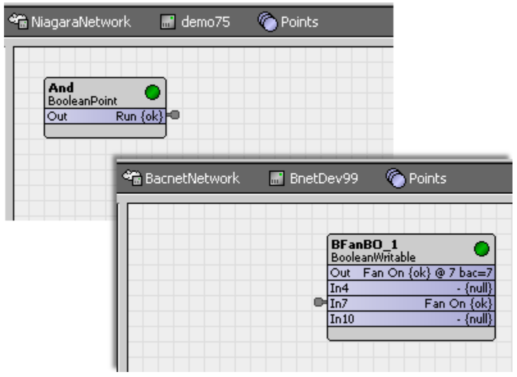
Figure 2-30 *Local object cannot link into Niagara proxy point*



In the example above, you cannot link into the Niagara proxy point "BFanBO_1."

So in the *other* (remote) station, you must make a Niagara proxy point for the And object, then link its output to the source point for "BFanBO_1." Figure 2-31 shows the wire sheet views with these objects.

Figure 2-31 *Source control object now a Niagara proxy point in remote station*



In a typical Supervisor station (serving primarily Px views), usually not much direct link control is needed, so this seldom applies. In addition, the schedule import/export mechanism allows for central scheduling control using local (imported) schedule components.

However, if you are engineering applications *between* JACE stations that require link control, or do have a use case where the Supervisor needs to globally write a value into the link logic in JACE stations, please understand you must always use Niagara proxy points in this fashion.

Note: *Avoid linking to* action *slots on Niagara proxy points, as such* links can be lost*. For more details, see the "Best practices for Niagara proxy points" section "Avoid links to proxy point actions" on page 2-25.*

NiagaraStation Alarms notes

To configure alarms in one station to be received in another station, you must add a **`StationRecipient`** in the AlarmService container of the "sending" (source) station. You then link whatever AlarmClass components are needed to that StationRecipient. For related details, refer to the sections "About the station recipient" and "About alarm class" in the *User Guide*.

It is not necessary to use the *same* AlarmClass components in the two stations (although that is one approach). In the receiving station (often, the Supervisor), if desired you can configure all alarms from a remote station to route to a single local AlarmClass. Or, you can also use a "prepend" or "append" scheme to route to different (but associated) AlarmClasses, where all schemes work based on the *names* of AlarmClasses. See "Prepend and Append alarm routing notes" for details.

Specify the alarm routing in the **`Alarms`** extension under the NiagaraStation that represents the remote JACE. See "Alarms extension properties" on page 1-35.

Note: *In the receiving station's AlarmService, if you want the remotely-generated alarms to appear in any alarm consoles, be sure to link associated AlarmClass components to the necessary AlarmConsole components.*

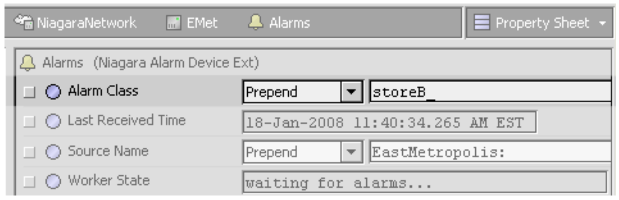
Prepend and Append alarm routing notes

The "Alarm Class" property of a NiagaraStation's **`Alarms`** extension offers two options: "Prepend" and "Append", with an associated text field for a text string. The typical application if for a Supervisor station that has *many* stations under its NiagaraNetwork, each using a "replicated" station database, meaning that each JACE station has identically-named AlarmClass components, along with other identical characteristics.

Consider a large "chain store" job where each JACE is installed at one retail store/building. Each JACE station has AlarmClasses named "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 could set the "Alarm Class" property to "Prepend" adding the store identifier ("storeA"_", "storeB_", etc.), as used when creating and naming AlarmClass components. See Figure 2-32 for an example property sheet.

Figure 2-32 *Example use of Prepend text in Alarm Class property of NiagaraStation Alarms extension*



Alarms (Niagara Alarm Device Ext)

Alarm Class	Prepend	storeB_
Last Received Time	18-Jan-2008 11:40:34.265 AM EST	
Source Name	Prepend	EastMetropolis:
Worker State	waiting for alarms...	

When a store alarm comes to this Supervisor, the "prepend text" is added to the originating AlarmClass in the JACE station, such that routing looks for that named AlarmClass, e.g. "storeB_humAlarm" or "storeW_runAlarm". This allows you to maintain the original alarm class mapping (at the "store station level") as well as segregate "by stores" at the Supervisor level.

Note: *This feature does not automatically create "virtual" alarm classes in the Supervisor; you still have to manually create all AlarmClass components needed. Also, check that AlarmClass component naming matches whatever "Prepend" or "Append" scheme is configured under the NiagaraStations in the Supervisor's NiagaraNetwork.*

Station Schedules import/export notes

Note: *Niagara schedule discovery was enhanced to permit operations that were previously blocked or unavailable. For details, see "Niagara Discover enhancements" on page 2-19.*

You configure sharing of NiagaraAX schedule components between stations from the "*receiving side*" station using the Niagara Schedule Import Manager. After you import each schedule component, a corresponding "schedule export descriptor" is automatically created in the "sending side" station. If necessary, you can review and adjust these export descriptors using the Niagara Schedule Export Edit dialog.

Under a NiagaraStation device, the Schedule Export Manager works uniquely from other drivers, so it is explained in more detail in the following sections.

- Niagara schedule import/export default configuration
- Schedule Export Edit
- Also see **"Schedule Import or Export "gang" edits" on page 1-52**

Niagara schedule import/export default configuration

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:

- **Receiving (slave) side:**
  Niagara schedule component with a ScheduleImportExt configured with "Execution Time" of *Manual*. The source schedule's configuration is imported ("pulled") only *once*, upon creation. If desired, you can manually Import it again.
- **Sending (master) side:**
  Corresponding schedule export descriptor with an "Execution Time" of *Interval*, at 5 minute rate. You can adjust this in the export descriptor using Schedule Export Edit.
  To review *all* properties of a schedule export descriptor, including status properties, you can view its property sheet—in the Nav tree under Schedules, simply double-click its Export icon.
  Note: *Default configuration does not mean the same schedule configuration is continuously exported ("pushed") to the remote schedule at this rate. Instead, the export descriptor keeps a "Subordinate Version" timestamp from the last export. If a configuration change occurs, the export descriptor compares the subordinate version time against the configured interval, and if necessary exports the schedule to the remote station.*

The default "master push" configuration is the most efficient (and recommended) way to keep imported schedules synchronized. However, if the stations are installed on a network configured such that this method is not allowed (perhaps firewall issues), you can configure things in reverse. This means configuring receiving side ScheduleImportExts with "Interval" execution times (to periodically "re-pull" schedule configuration), and set corresponding schedule export descriptors to be disabled.

Schedule Export Edit

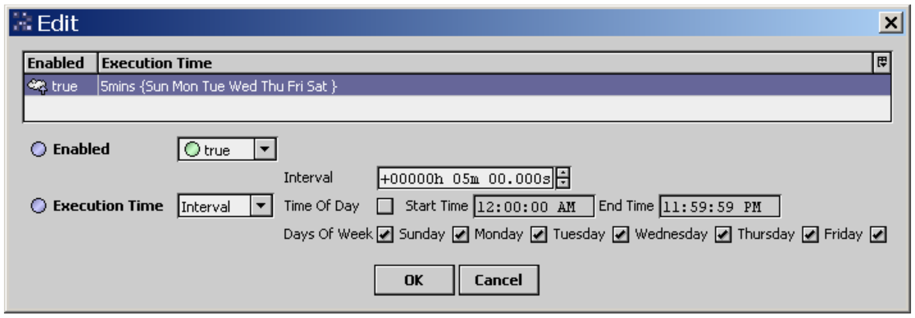
In the Schedule Export Manager, you adjust any schedule export descriptor(s) shown by selecting (highlighting) it and clicking the Edit button at the bottom of the view (or on the toolbar).

Note: *Unlike in other Manager views, a double-click on a descriptor is not the same as Edit. Instead, double-click provides the Scheduler view for that exported schedule component. This can be useful to review and if necessary, make changes to its configuration. For related details, refer to the section "Schedule component views" in the* User Guide.

Edit

The Edit dialog appears with the schedule export descriptor listed (Figure 2-33).

Figure 2-33 *Edit dialog in Schedule Export Manager*



Edit

Enabled	Execution Time
true	5mins {Sun Mon Tue Wed Thu Fri Sat }

Enabled: true

Execution Time: Interval

Interval +00000h 05m 00.000s

Time Of Day ☐ Start Time 12:00:00 AM End Time 11:59:59 PM

Days Of Week ✓ Sunday ✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓

OK Cancel

Note: *The Edit dialog shows some configuration properties of the schedule export descriptor. To access all properties, (including all status properties) go to its property sheet. Note that if you double-click the Nav tree icon for an export descriptor, its property sheet displays.*

The following related topics also apply:

- Niagara Schedule Export properties
- Niagara schedule import/export default configuration
- Also see "Manager table features" on page 1-21

Niagara Schedule Export properties

Properties in the Edit dialog of a schedule export descriptor are:

- **Enabled**
  By default true. While set to false (export descriptor *disabled*), export connections are not attempted to update the remote (imported) schedule.
- **Execution Time**
  Determines when an export update is made to the remote (imported) schedule, *providing that a configuration change occurred in the local schedule that requires synchronization (export)*. For more details, see "Niagara schedule import/export default configuration" on page 2-28.Options are either Daily, Interval (default), or Manual. If `Manual`, the following properties are unavailable:
  - **Time of Day (Daily)**
    Configurable to any daily time. Default is 2:00am.
  - **Randomization (Daily)**
    When the next execution time calculates, a random amount of time between zero milliseconds and this interval is added to the Time of Day. May prevent "server flood" issues if too many schedule exports are executed at the same time. Default is zero (no randomization).
  - **Days of Week (Daily and Interval)**
    Select (check) days of week for archive execution. Default is all days of week.
  - **Interval (Interval)**
    Specifies repeating interval for export execution. Default is every 15 minutes.
  - **Time of Day (Interval)**
    Specifies start and end times for interval. Default is 24-hours (start 12:00am, end 11:59pm).

Niagara histories notes

Creating Niagara history import and history export descriptors is how you save a Niagara history to a different location (station) from where it originated. In a typical application, this is considered *archiving*. For example, an originating history (with a limited record count) may be in a JACE station. If *imported* to a supervisor station, its history import descriptor can be configured such that the imported history in the supervisor has "unlimited" record capacity. The JACE history can run collecting only the last 500 records, while the imported history in the supervisor will collect all (unlimited) records.

This section explains items unique to working with histories under a NiagaraNetwork.

This includes the **`Niagara History Export Manager`** view on a NiagaraStation's Histories device extension, as well as Niagara history features starting in AX-3.5. See the following main sections for details:

- NiagaraStation Histories features
- Niagara History Import Manager
- Niagara History Export Manager
- On demand properties in history import descriptors

NiagaraStation Histories features

Note: *Niagara history discovery was enhanced to permit operations that were previously blocked or unavailable. For details, see "Niagara Discover enhancements" on page 2-19.*

Niagara history features that affect NiagaraNetwork configuration include:

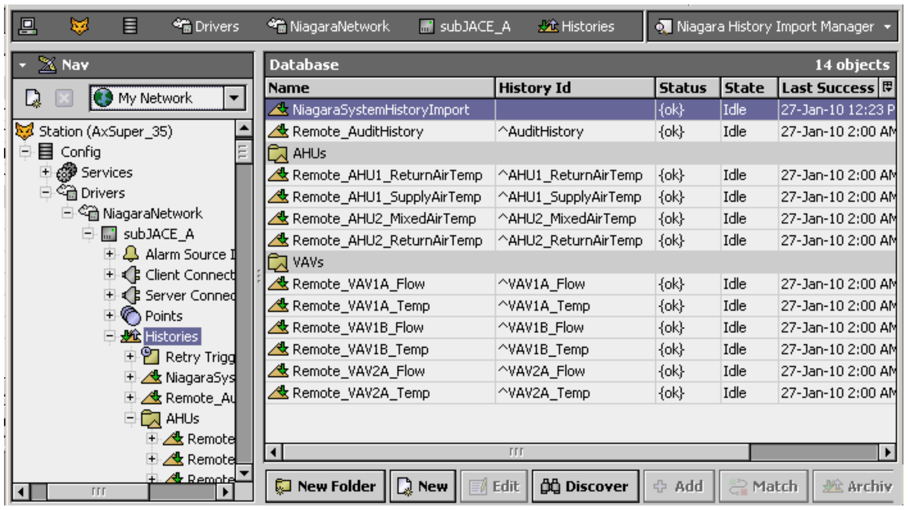
- Starting in AX-3.5, it is possible to use components known as "export tags" in a JACE station, that upon a "Join" command, will *automatically add* pre-configured Niagara HistoryImport descriptors in a remote *Supervisor* station, and import those JACE histories. This is one of many functions provided by export tags. For complete details, see the document *NiagaraAX Export Tags*.
- Starting in AX-3.5, you can create special *folders* under the **`Histories`** extension of a NiagaraStation, using a "**`New Folder`**" button in the Niagara History Import Manager or History Export Manager, to better organize history import and export descriptors. Each folder provides manager views.
- Starting in AX-3.5, a "**`Device Histories View`**" is on a NiagaraStation's Histories extension, which provides shortcuts to histories. Although not unique to the NiagaraNetwork, this feature may be useful, especially in a Supervisor station. See "About the Device Histories View" on page 1-49.
- "On demand" polling is also available for both local histories and imported histories. On any history enabled for this, a system user can now poll for live data when viewing its history chart or history table view, by clicking the "Live Updates" toggle button (play icon). Refer to the *NiagaraAX User Guide* section "About On Demand history views" for related details.
- You can also import/export histories based upon text "system tag" patterns, versus explicit History Ids (i.e. "Discover" in the History Import (or Export) Manager, then selecting specific histories to add). This utilizes a "System Tags" property in history extensions. Refer to the *NiagaraAX User Guide* section "Configure history extensions" for related details.

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

Niagara History Import Manager

The Niagara History Import Manager is the default view on a NiagaraStation's **`Histories`** extension (Figure 2-34). It offers standard features as described in "History Import Manager" on page 1-46.

Figure 2-34 *History Import Manager under a NiagaraStation*



Database 14 objects

Name	History Id	Status	State	Last Success
NiagaraSystemHistoryImport		{ok}	Idle	27-Jan-10 12:23 P
Remote_AuditHistory	^AuditHistory	{ok}	Idle	27-Jan-10 2:00 AM
AHUs				
Remote_AHU1_ReturnAirTemp	^AHU1_ReturnAirTemp	{ok}	Idle	27-Jan-10 2:00 AM
Remote_AHU1_SupplyAirTemp	^AHU1_SupplyAirTemp	{ok}	Idle	27-Jan-10 2:00 AM
Remote_AHU2_MixedAirTemp	^AHU2_MixedAirTemp	{ok}	Idle	27-Jan-10 2:00 AM
Remote_AHU2_ReturnAirTemp	^AHU2_ReturnAirTemp	{ok}	Idle	27-Jan-10 2:00 AM
VAVs				
Remote_VAV1A_Flow	^VAV1A_Flow	{ok}	Idle	27-Jan-10 2:00 AM
Remote_VAV1A_Temp	^VAV1A_Temp	{ok}	Idle	27-Jan-10 2:00 AM
Remote_VAV1B_Flow	^VAV1B_Flow	{ok}	Idle	27-Jan-10 2:00 AM
Remote_VAV1B_Temp	^VAV1B_Temp	{ok}	Idle	27-Jan-10 2:00 AM
Remote_VAV2A_Flow	^VAV2A_Flow	{ok}	Idle	27-Jan-10 2:00 AM
Remote_VAV2A_Temp	^VAV2A_Temp	{ok}	Idle	27-Jan-10 2:00 AM

Note: *A "***`New Folder`***" button is available in this view for adding "archive folders", to help organize history import (or export) descriptors. Each folder has its own history manager views.*

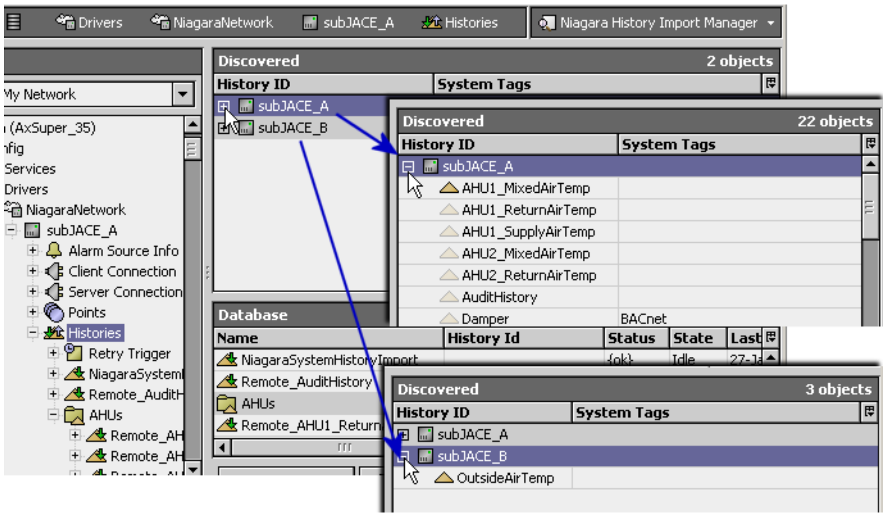
The following sections provide more details unique to Niagara history imports:

- Discovered selection notes
- Niagara History Import properties
- On demand properties in history import descriptors
- Using System Tags to import Niagara histories

Discovered selection notes

In the Niagara History Import Manager, discovered histories of a NiagaraStation are under an expandable *tree structure*, organized by station name (Figure 2-35).

Figure 2-35 *Expand Niagara stations to see all Niagara histories*



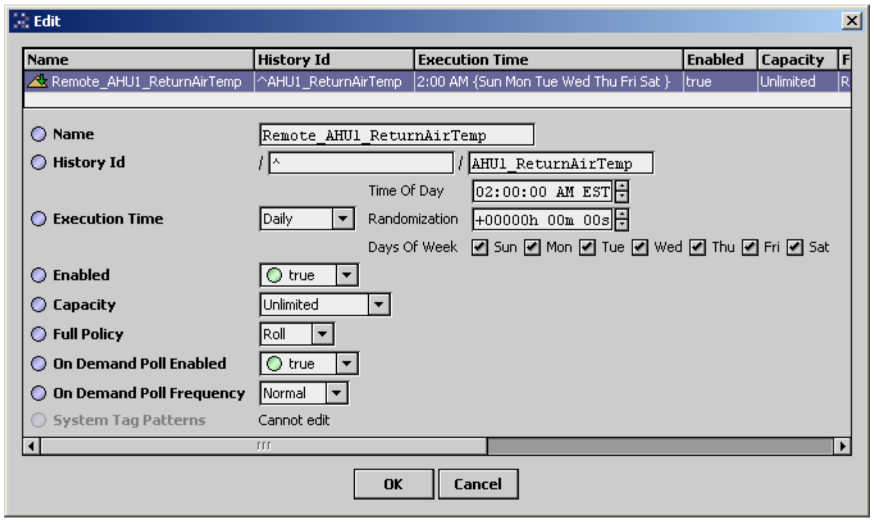
Histories under the *same* station name as the parent NiagaraStation (device) component are *local histories* for that station. Histories under any other stations represent histories currently imported into (or exported *to*) that station.

For example, discovered histories in Figure 2-35 for NiagaraStation subJACE_A include local histories (expanded, top); another "imported" history from remote station subJACE_B is shown below.

Note: *From any NiagaraStation, you can import both its "local" histories and already-imported histories, as needed. However, unless circumstances warrant a "relay archive method," it may be best to import histories directly from the source station whenever possible.*

Niagara History Import properties

Figure 2-36 *Edit dialog for Niagara HistoryImport descriptor*



Name	History Id	Execution Time	Enabled	Capacity	F
Remote_AHU1_ReturnAirTemp	^AHU1_ReturnAirTemp	2:00 AM {Sun Mon Tue Wed Thu Fri Sat }	true	Unlimited	R

Name	Remote_AHU1_ReturnAirTemp
History Id	/ ^ / AHU1_ReturnAirTemp
Execution Time	Daily; Time Of Day 02:00:00 AM EST; Randomization +00000h 00m 00s; Days Of Week ✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true
Capacity	Unlimited
Full Policy	Roll
On Demand Poll Enabled	true
On Demand Poll Frequency	Normal
System Tag Patterns	Cannot edit

Properties of Niagara History Import descriptors available in the `Edit` or `Add` dialog are as follows:

- **Name**
  Name for the history import descriptor component. If discovered, typically left at default.
  Note: *Editing name does not affect name of the resulting history (imported into station).*
- **History Id**
  This property specifies the history name in the local station's history space, using two parts: "/<*stationName*>" and "/<*historyName*>". If learned, station name is "^" (see note below) and history

name reflects the source history name. Typically, you leave both fields at default values, or edit the second (<historyName>) field only.
Note: *The "^" character is basically a shorthand character to refer to the device name of the parent container (NiagaraStation component). This may be useful if you have multiple JACEs with histories named the same. You can create and configure a single History Import Descriptor and then duplicate and paste it under the other stations without having to go in and change the station name each time.*
Note for a Niagara System History Import descriptor, the History Id property is not applicable.

- **Execution Time**
  Either Daily (default), Interval, or Manual. If Manual, properties below are not available:
  - **Time of Day (Daily)**
    Configurable to any daily time. Default is 2:00am.
  - **Randomization (Daily)**
    When the next execution time calculates, a random amount of time between zero milliseconds and this interval is added to the Time of Day. May prevent "server flood" issues if too many history archives are executed at the same time. Default is zero (no randomization).
  - **Days of Week (Daily and Interval)**
    Select (check) days of week for archive execution. Default is all days of week.
  - **Interval (Interval)**
    Specifies repeating interval for archive execution. Default is every 15 minutes.
  - **Time of Day (Interval)**
    Specifies start and end times for interval. Default is 24-hours (start 12:00am, end 11:59pm).
- **Enabled**
  Default is true. If set to false, does not execute import of history data.
- **Capacity**
  Specifies local storage capacity for imported history, either as Unlimited or Record Count.
  If set to Record Count, the following property is available:
  - **records**
    Number of records (samples) to locally store. Default is 500.
- **Full Policy**
  Either Roll (default) or Stop. Applies only if capacity is set to record count
  - If Roll, upon record count, oldest records become overwritten by newest records.
  - If Stop, upon record count, importing stops (until history records are locally deleted).
- **On Demand Poll Enabled**
  Either true (default) or false. See the next section "On demand properties in history import descriptors" for more details on this property and the related poll frequency property.
- **On Demand Poll Frequency**
  Either "Fast", "Normal", or "Slow".
- **System Tag Patterns**
  Specifies one or more text strings matched against text values in "System Tags" properties of remote history extensions, where matching text patterns result in histories imported into the local history space. For more details, see "Using System Tags to import Niagara histories" on page 2-32.

On demand properties in history import descriptors

When using the Niagara History Import Manager to import remote histories, there are two properties in a NiagaraHistoryImport descriptor related to "On demand" polling:

- **On Demand Poll Enabled**
  Either true (default) or false.
  - If true, a system user will be able to use the "Live Updates" (play) button in history views to poll for live data for the associated imported history(ies).
  - If false, this button will not be available in history views for the associated imported history(ies).
- **On Demand Poll Frequency**
  Either "Fast", "Normal", or "Slow", which references the "On Demand Poll Scheduler" rates under the NiagaraNetwork's "History Policies" container slot. See "About History Policies" on page 2-7 for related details.

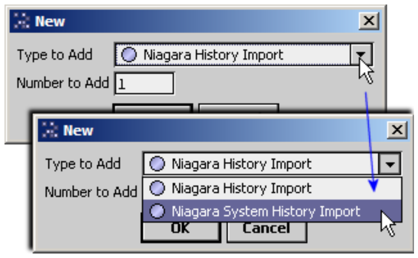
Using System Tags to import Niagara histories

"System Tags" support is available in the Niagara History Import Manager. This provides an alternate way to import histories from remote NiagaraStations, using "Niagara *system* history import descriptors". This allows you to import many histories at once, instead of individually importing histories "one at a time."

For this to work, remote history extensions (for histories to be imported) must be configured with "System Tags" property values. Refer to the *NiagaraAX User Guide* section "Configure history extensions" for related details.

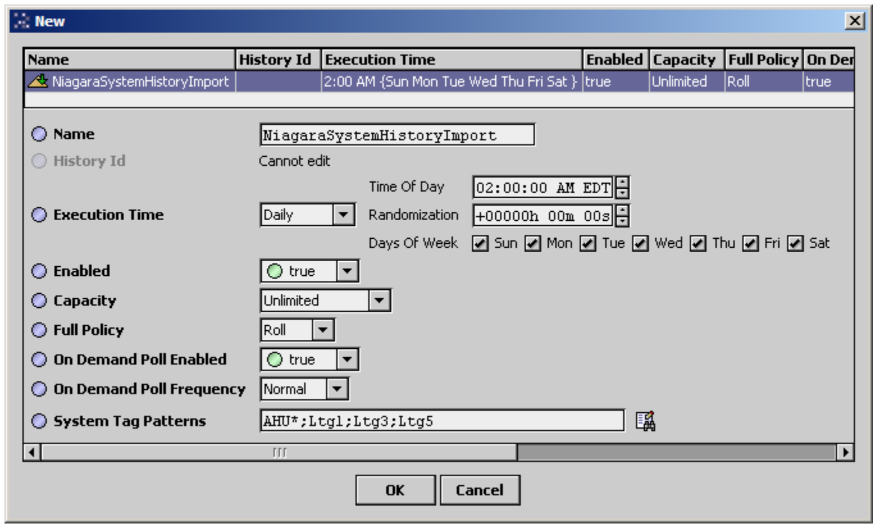
Instead of using the import manager's "Learn Mode" (Discover) to add history import descriptors, you *manually add* new descriptors in the Niagara History Import Manager, by selecting `NiagaraSystem-HistoryImport` in the "Type" drop-down control of the New popup dialog, as shown in Figure 2-37.

Figure 2-37 *Add New Niagara System History Import descriptors for System Tags feature*



In the resulting **New** dialog for the system history export descriptors, you enter one or more "System Tags" as a text pattern, as shown in Figure 2-38.

Figure 2-38 *Enter "System Tags Pattern" text to match remote history extensions' "System Tags" values*



New

Name	History Id	Execution Time	Enabled	Capacity	Full Policy	On Der
NiagaraSystemHistoryImport		2:00 AM {Sun Mon Tue Wed Thu Fri Sat }	true	Unlimited	Roll	true

Name	NiagaraSystemHistoryImport
History Id	Cannot edit
Execution Time	Daily; Time Of Day 02:00:00 AM EDT; Randomization +00000h 00m 00s; Days Of Week ✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true
Capacity	Unlimited
Full Policy	Roll
On Demand Poll Enabled	true
On Demand Poll Frequency	Normal
System Tag Patterns	AHU*;Ltg1;Ltg3;Ltg5

You can enter *multiple* System Tags in this "System Tag Patterns" property, using a semicolon (`;`) as delimiter between System Tags. In addition, you can use an asterisk (`*`) as a "wildcard" when entering System Tag values. In the Figure 2-38 example, the System Tag Patterns value is:

```
AHU*;Ltg1;Ltg3;Ltg5
```

In this example, remote points with history extensions configured with System Tags property values including an entry beginning with "AHU" and/or "Ltg1", "Ltg2", or "Ltg3" will be imported after this is added and executed. For example, history extensions with a System Tags property value of "AHU1", AHU_1", "AHU 1", "AHU2", and so will be included, as well as those including "Ltg1", "Ltg3" and "Ltg5", but ones with a System Tags value of only "Ahu1", "Ltg2", or "Ltg 1" will not.

Note that other properties like Execution Time, Capacity, Full Policy, and On Demand Poll Enabled apply to the associated imported histories, and operate the same as for regular HistoryImport descriptors. See "Niagara History Import properties" on page 2-31 for further details.

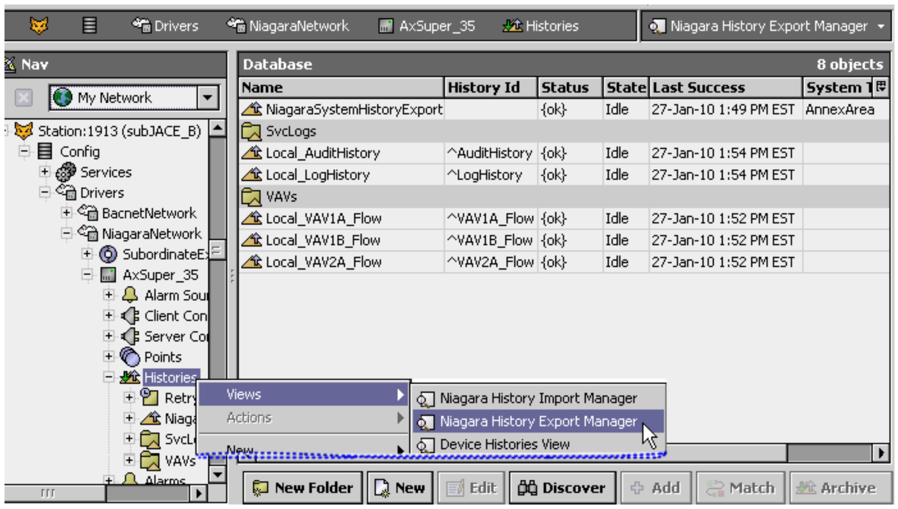
Niagara History Export Manager

A History Export Manager view is available (only) on a NiagaraStation's *Histories* extension, apart from "database device" components in some RdbmsNetwork drivers (see the *Rdbms Driver Guide* for details).

Note: *If using the Bacnet driver, Niagara histories in the local station can also be "exposed" to all networked BACnet devices as BACnet "Trend Log objects". However, this is done using a different view under the Local Device component in the BacnetNetwork. See the* BACnet Guide *for more details.*

Like other managers, the **`Niagara History Export Manager`** is a table-based view (Figure 2-39).

Figure 2-39 *History Export Manager under a NiagaraStation*



Database 8 objects

Name	History Id	Status	State	Last Success	System T
NiagaraSystemHistoryExport		{ok}	Idle	27-Jan-10 1:49 PM EST	AnnexArea
SvcLogs					
Local_AuditHistory	^AuditHistory	{ok}	Idle	27-Jan-10 1:54 PM EST	
Local_LogHistory	^LogHistory	{ok}	Idle	27-Jan-10 1:54 PM EST	
VAVs					
Local_VAV1A_Flow	^VAV1A_Flow	{ok}	Idle	27-Jan-10 1:52 PM EST	
Local_VAV1B_Flow	^VAV1B_Flow	{ok}	Idle	27-Jan-10 1:52 PM EST	
Local_VAV2A_Flow	^VAV2A_Flow	{ok}	Idle	27-Jan-10 1:52 PM EST	

Each row typically represents a history *export descriptor*. Each descriptor specifies how data from a *local history* is exported ("pushed") from the station to a selected NiagaraStation, where it appears as a history.

Note: *A "**New Folder**" button is available in this view for adding "archive folders", to help organize history export (or import) descriptors. Each folder has its own history manager views.*

You use this view to create, edit, and delete history *export* descriptors. Each export descriptor you add results in the creation of a Niagara history on that remote station.

Following station configuration, this view provides a status summary for exporting local histories. You can also use it to issue manual "Archive" commands to one or more history descriptors. This causes an export "push" of history data into the selected histories at the remote Niagara station.

Note: *Only history export descriptors appear in the History Export Manager view—any other components that may also reside under Histories do not appear. For example, you do not see the default "Retry Trigger" component (see "About the Retry Trigger" on page 1-34), or history import descriptors. However, you can use the Histories property sheet or the Nav tree to access these items.*

The "capacity" and "fullPolicy" configuration for each "exported" history (as created on the remote station), is determined by that target station's NiagaraNetwork component's "History Policies." Before exporting any histories, you should review and adjust these policies as needed. For more details, see "About History Policies" on page 2-7.

At the bottom of the view, the button "**New**" lets you manually create new export descriptors in the station. An "**Edit**" button lets you edit one or more export descriptors. Buttons "**Discover**," "**Add**" and "**Match**" are also available, (these work similarly as in the Point Manager). An "**Archive**" button is available to manually export (push data) into one or more selected histories.

For more details, see:

- Niagara History Export New
- Niagara History Export Edit
- About History Export Discover and Match (Learn Process)
- Using System Tags to export Niagara histories

Niagara History Export New

Button **New** exists in a NiagaraStation's History Export Manager view, but is typically used only if:

- Using "system tags" to export local histories to the target remote NiagaraStation, versus using online discovery and selecting individual histories. For more details, see **"Using System Tags to export Niagara histories" on page 2-37**.
- Engineering offline. If offline, Match may be used later (when online with the device).

Note: *A New tool is also on the Niagara History Export Manager toolbar, and in the Manager menu.*

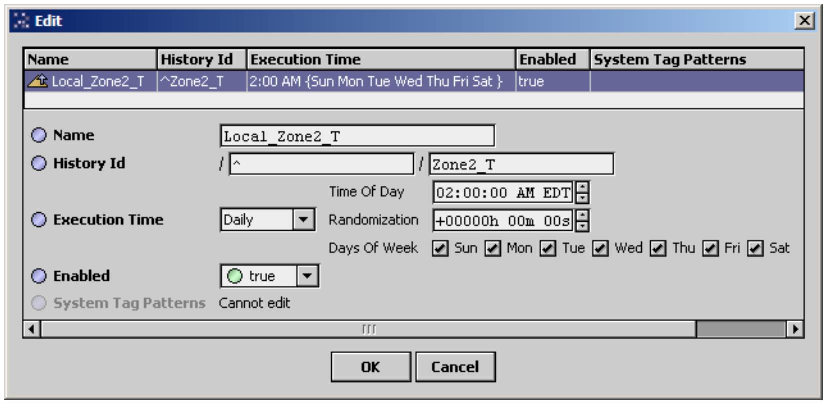
Niagara History Export Edit

In the Niagara History Export Manager, you can Edit any history export descriptor (or system history export descriptor) in the station database by simply double-clicking it.

Edit

The Edit dialog appears with the export descriptor listed (Figure 2-40).

Figure 2-40 *Edit dialog in History Export Manager (single history)*



Edit

Name	History Id	Execution Time	Enabled	System Tag Patterns
Local_Zone2_T	^Zone2_T	2:00 AM {Sun Mon Tue Wed Thu Fri Sat }	true	

Name	Local_Zone2_T	
History Id	/ ^ / Zone2_T	
Execution Time	Daily	Time Of Day: 02:00:00 AM EDT
		Randomization: +00000h 00m 00s
		Days Of Week: ✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true	
System Tag Patterns	Cannot edit	

Note: *The Edit dialog shows configuration properties of the history export descriptor, plus Name (equivalent to the right-click Rename command on the descriptor). To access all properties, (including all status properties) go to its property sheet.*

The following section explains Niagara History Export properties:

Niagara History Export properties

Properties of Niagara history export descriptors available in the Edit or Add dialog are as follows:

- **Name**
  Name for history export descriptor component. Typically left at default. Begins with "`Local_`" for any history originating from the local station. Or if adding a NiagaraSystemHistoryExport, defaults to "`NiagaraSystemHistoryExport`" (appending numerals as needed to keep unique).
  Note: *Editing name does not affect name of resulting history (exported into remote station).*
- **History Id**
  For a NiagaraHistoryExport descriptor, this property specifies the history name to be created in the remote station's history space, using two parts: "/<*stationName*>" and "/<*historyName*>". Histories originating in the local station show a "^" (shorthand for local station name), and history name reflects the source history name. Typically, you leave both fields at default values.
  Note: *For a NiagaraSystemHistoryExport descriptor, the History Id property is not applicable.*
- **Execution Time**
  Either Daily (default), Interval, or Manual. If Manual, properties below are not available:
  - **Time of Day (Daily)**
    Configurable to any daily time. Default is 2:00am.
  - **Randomization (Daily)**
    When the next execution time calculates, a random amount of time between zero milliseconds and this interval is added to the Time of Day. May prevent "server flood" issues if too many history archives are executed at the same time. Default is zero (no randomization).
  - **Days of Week (Daily and Interval)**
    Select (check) days of week for archive execution. Default is all days of week.
  - **Interval (Interval)**
    Specifies repeating interval for archive execution. Default is every 15 minutes.
  - **Time of Day (Interval)**
    Specifies start and end times for interval. Default is 24-hours (start 12:00am, end 11:59pm).
- **Enabled**
  Default is true. If set to false, history data is not exported.
- **System Tag Patterns**
  (Modifiable only for a Niagara*System*HistoryExport descriptor) Specifies one or more text strings matched against text values in "System Tags" properties of local history extensions, where matching text patterns result in histories exported into the remote history space. For more details, see "Using System Tags to export Niagara histories" **on page 2-37**.

Note: *The capacity and full policy of any exported history (created on the remote station) is determined by rules under that station's NiagaraNetwork "History Policies," and is set at creation time only. For details, see "About History Policies" on page 2-7.*

About History Export Discover and Match (Learn Process)

Unless working offline, you can use the learn process to export histories in the station. As with other NiagaraAX learns, this is a two-step *process* in the Niagara History Export Manager, where you:

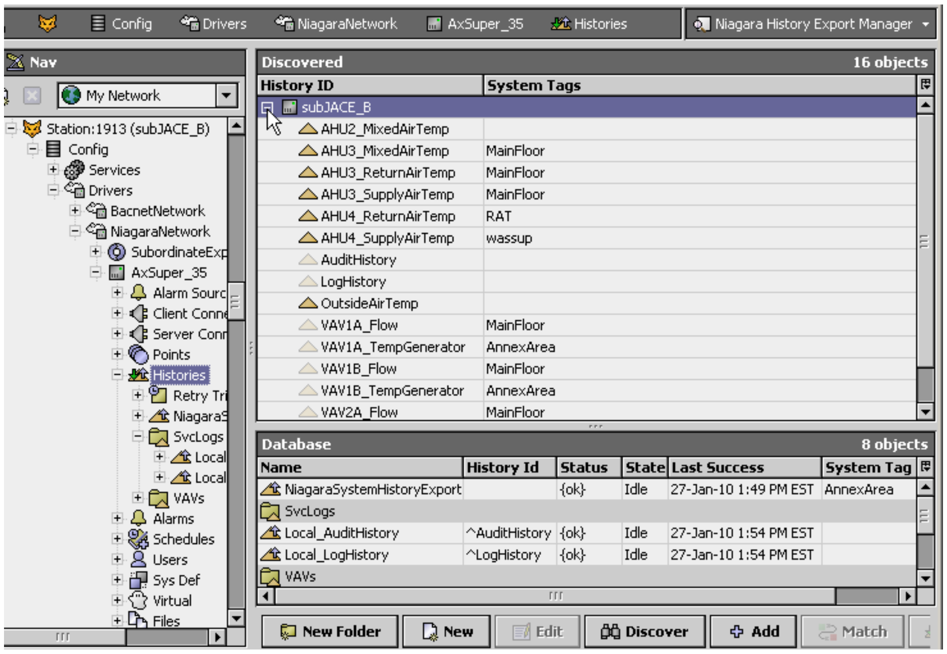
1. Under a selected NiagaraStation, use its (Histories extension) Niagara Histories Export Manager view to Discover (local) station histories as *candidates* for export to that station as histories.
2. Select and Add from those histories, creating history export descriptors under the station's Histories container.

Note: *The Histories Export Manager reinforces this process by providing two separate panes in the view whenever you enter "Learn Mode." See "About Learn toggle" on page 1-18.*

Discover

When you click Discover, the Histories Export Manager splits into two panes, or *Learn Mode* (Figure 2-41). The top discovered pane is a collapsed *tree structure* of all Niagara histories of the local station. Click to expand and select histories for export. See "Discovered selection notes" on page 2-37 for more details.

Figure 2-41 *Discover splits Niagara History Export Manager*



Discovered — 16 objects

History ID	System Tags
subJACE_B	
AHU2_MixedAirTemp	
AHU3_MixedAirTemp	MainFloor
AHU3_ReturnAirTemp	MainFloor
AHU3_SupplyAirTemp	MainFloor
AHU4_ReturnAirTemp	RAT
AHU4_SupplyAirTemp	wassup
AuditHistory	
LogHistory	
OutsideAirTemp	
VAV1A_Flow	MainFloor
VAV1A_TempGenerator	AnnexArea
VAV1B_Flow	MainFloor
VAV1B_TempGenerator	AnnexArea
VAV2A_Flow	MainFloor

Database — 8 objects

Name	History Id	Status	State	Last Success	System Tag
NiagaraSystemHistoryExport		{ok}	Idle	27-Jan-10 1:49 PM EST	AnnexArea
SvcLogs					
Local_AuditHistory	^AuditHistory	{ok}	Idle	27-Jan-10 1:54 PM EST	
Local_LogHistory	^LogHistory	{ok}	Idle	27-Jan-10 1:54 PM EST	
VAVs					

In Learn Mode, the two panes in the Histories Export Manager operate as follows:

- **Discovered (top pane)**
  Lists all histories in the *local* station (station you are engineering).
- **Database (bottom pane)**
  Lists history export descriptors and history *system* export descriptors currently in the station database.
  - Each NiagaraHistoryExport descriptor has a "one-to-one" associated history, exported to that remote station.
  - Each Niagara*System*HistoryExport descriptor may, and often does, result in many associated exported histories in the remote station. Note in the database pane you can spot these export descriptors by the *blank* History Id value, as well as a text value in the "System Tag Patterns" column.

Note: *As necessary, drag the border between the two panes to resize. Also (at any time), toggle between the two-pane Learn Mode and the single-pane (Database) view by clicking the Learn Mode tool in the toolbar, or using the Learn Mode command in the Manager menu.*

Add The Add button is available in Learn Mode when you have one or more items selected (highlighted) in the top discovered pane. When you click Add, a dialog appears that allows you to edit properties before the history export descriptor(s) are created in the station.

The Add dialog is identical to the history export descriptor Edit dialog. For details on properties, see "Niagara History Export properties" on page 2-35.

Discovered selection notes

In the **`Niagara History Export Manager`**, discovered local histories are under an expandable *tree structure*, organized by station name (see Figure 2-41 on page 36). Histories under the *same* station name as the local station are originated in that local station. Histories under any *other* station nodes represent histories currently imported (or exported) into the local station.

For example, discovered histories in Figure 2-41 for local station "subJACE_B" include locally-originated histories (at top); any other histories originating from other stations would be under a different expandable node (not shown).

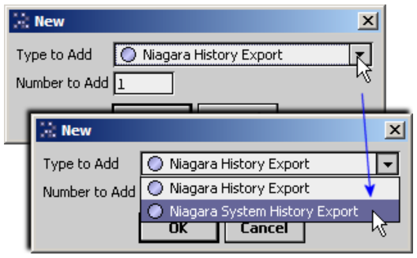
Note: *To any NiagaraStation, you can export both a station's "locally-originated" histories as well as already-imported histories, as needed. However, unless circumstances warrant a "relay archive method," it may be best to export only "locally-originated" histories.*

Using System Tags to export Niagara histories

In the Niagara History Export Manager you can use "NiagaraSystemHistoryExport" descriptors to export locally-sourced histories. This allows you to export many histories at once, instead of individually exporting them "one at a time." For this to work, local history extensions (for histories to be exported) must be configured with "System Tags" property values. Refer to the *NiagaraAX User Guide* section "Configure history extensions" for related details.

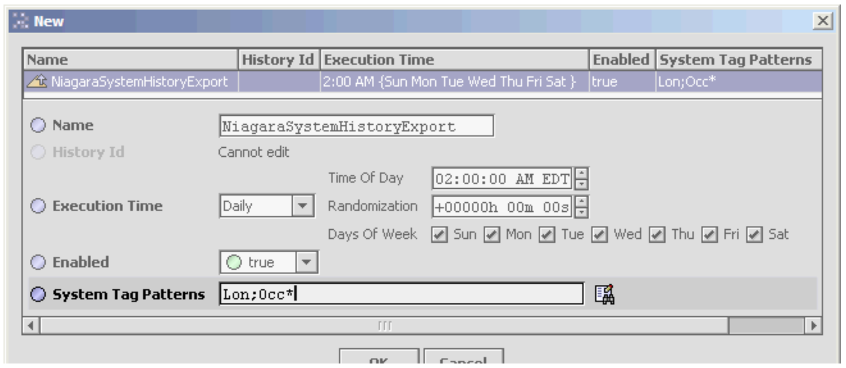
Instead of using the export manager's "Learn Mode" (Discover) to add history export descriptors, you *manually add* new descriptors in the Niagara History Export Manager, by selecting `NiagaraSystem-HistoryExport` in the "Type" drop-down control of the New popup dialog, as shown in Figure 2-42.

Figure 2-42 *Add New Niagara System History Export descriptors for System Tags feature*



In the resulting New dialog for the system history export descriptors, you enter one or more "System Tags" as a text pattern, as shown in Figure 2-43.

Figure 2-43 *Enter "System Tags Pattern" text to match local history extensions' "System Tags" values*



New

Name	History Id	Execution Time	Enabled	System Tag Patterns
NiagaraSystemHistoryExport		2:00 AM {Sun Mon Tue Wed Thu Fri Sat }	true	Lon;Occ*

Name	NiagaraSystemHistoryExport
History Id	Cannot edit
Execution Time	Daily; Time Of Day 02:00:00 AM EDT; Randomization +00000h 00m 00s; Days Of Week ✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true
System Tag Patterns	Lon;Occ*

You can enter *multiple* System Tags in this "System Tag Patterns" property, using a semicolon (`;`) as delimiter between System Tags. In addition, you can use an asterisk (`*`) as a "wildcard" when entering System Tag values. In the Figure 2-43 example, the System Tag Patterns value is:

```
    Lon;Occ*
```

In this example, any local points with history extensions configured with System Tags property values of "Lon" and/or a value beginning with "Occ" will be exported after this is added and executed.

For example, history extensions with a System Tags property value of "Lon", "Occupancy" or "OccUnocc" will be included, but ones with a System Tags value of only "OcUnoc" or "Lon1" will not. Note that the Execution Time property values apply to all associated exports, and operate the same as for regular NiagaraHistoryExport descriptors. See "Niagara History Export properties" on page 2-35.

About Niagara virtual components

Niagara virtuals provide an attractive *alternative* to creating Niagara proxy points when engineering a Supervisor station—especially for general Px page presentation. With enhancements starting in AX-3.5 and continued in AX-3.7, it is expected that this feature should dramatically reduce the general need for Niagara *proxy points*. Niagara virtuals are also an integral part in the "PxViewTag" *export tag* process—although in that case the creation of them is basically "automatic".

Prior to Niagara virtuals, accessing any real-time data that originates in *another* NiagaraAX station required creating Niagara proxy points, under the Points extension of an associated NiagaraStation component. The usual "large scale" application for Niagara proxy points was in a Supervisor station, where values in remote JACE stations could be modeled, then be graphically presented on Px pages hosted by the Supervisor. A station in a JACE may also have a few Niagara proxy points, typically in cases where remote data is required within its control logic. However, Niagara proxy points have historically been synonymous with a Supervisor. For more details on Niagara proxy points, see "Niagara proxy point notes" on page 2-24.

Niagara virtuals background

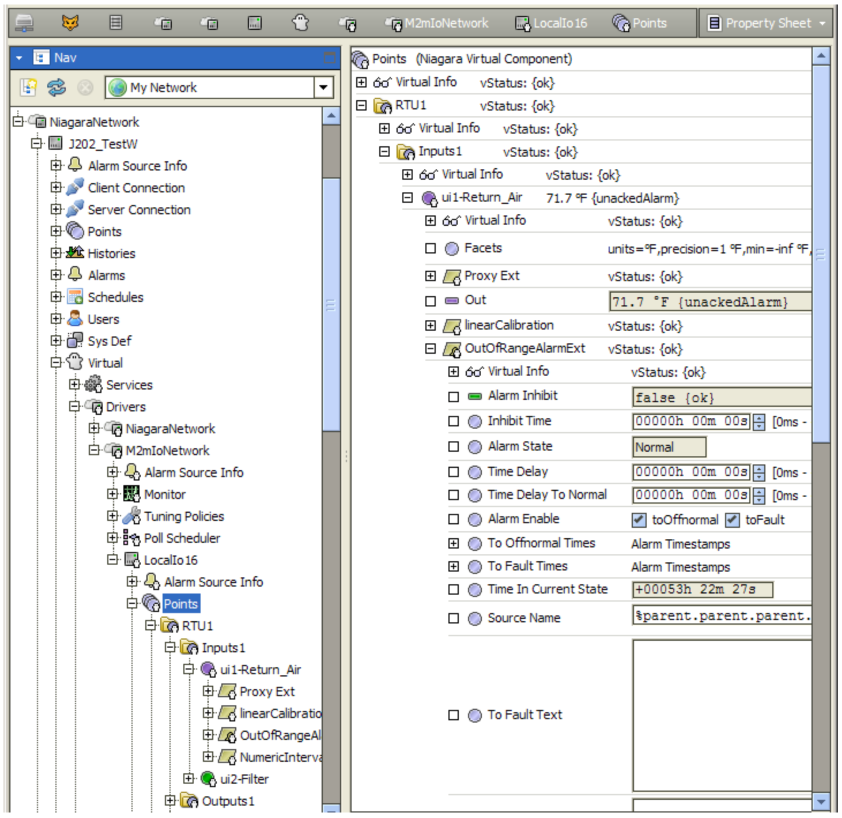
Starting in AX-3.4, Baja virtual components were implemented in the niagaraDriver. This allowed a Supervisor station access to Niagara virtual components (or simply "Niagara virtuals"), via a "Virtual gateway" under each NiagaraStation in its NiagaraNetwork. These dynamically-created *Niagara virtual components* provided monitor access to the *entire component structure* of each remote JACE station—without the overhead (or engineering necessity) of Niagara proxy points.

In AX-3.5, enhancements were made to Niagara virtuals. Most notable was that Niagara virtuals now provided *write access* to most properties. In addition, point status was improved for virtuals of control points (typical usage in proxy points among various driver types).

Niagara virtuals in AX-3.7

Starting in AX-3.7, *more* enhancements were made to Niagara virtuals, where the former "Virtual gateway" (Virtual) component of a NiagaraStation is now a "NiagaraVirtualDeviceExt", sourced from a separate `niagaraVirtual` module. Figure 2-44 shows example virtual components under the Virtual device extension of a NiagaraStation.

Figure 2-44 *Expanded VirtualDeviceExt of a NiagaraStation in a Supervisor station's NiagaraNetwork*



```
NiagaraNetwork
  J202_TestW
    Alarm Source Info
    Client Connection
    Server Connection
    Points
    Histories
    Alarms
    Schedules
    Users
    Sys Def
    Virtual
      Services
      Drivers
        NiagaraNetwork
        M2mIoNetwork
          Alarm Source Info
          Monitor
          Tuning Policies
          Poll Scheduler
          LocalIo16
            Alarm Source Info
            Points
              RTU1
                Inputs1
                  ui1-Return_Air
                    Proxy Ext
                    linearCalibratio
                    OutOfRangeAl
                    NumericInterva
                  ui2-Filter
                Outputs1
```

```
Points  (Niagara Virtual Component)
Virtual Info            vStatus: {ok}
RTU1                    vStatus: {ok}
  Virtual Info          vStatus: {ok}
  Inputs1               vStatus: {ok}
    Virtual Info        vStatus: {ok}
    ui1-Return_Air      71.7 °F {unackedAlarm}
      Virtual Info          vStatus: {ok}
      Facets                units=°F,precision=1 °F,min=-inf °F,
      Proxy Ext             vStatus: {ok}
      Out                   71.7 °F {unackedAlarm}
      linearCalibration     vStatus: {ok}
      OutOfRangeAlarmExt    vStatus: {ok}
        Virtual Info            vStatus: {ok}
        Alarm Inhibit           false {ok}
        Inhibit Time            00000h 00m 00s  [0ms -
        Alarm State             Normal
        Time Delay              00000h 00m 00s  [0ms -
        Time Delay To Normal    00000h 00m 00s  [0ms -
        Alarm Enable            ✓ toOffnormal  ✓ toFault
        To Offnormal Times      Alarm Timestamps
        To Fault Times          Alarm Timestamps
        Time In Current State   +00053h 22m 27s
        Source Name             %parent.parent.parent.
        To Fault Text
```

In AX-3.7, the basic operation of Niagara virtuals remains unchanged from AX-3.6 and AX-3.5. As shown in Figure 2-44, a small "virtual" ghost ( ) is superimposed in the lower right over the "normal" icon for each Niagara virtual component representing any component type—a visual reminder that you are looking into the "virtual component space" representing that station.

However, the AX-3.7 and later virtual architecture provides the following benefits:

- "Niagara virtuals to Niagara virtuals" are now supported. For example, in an "*n*-tiered" system, a top-tiered Supervisor station can map Niagara virtuals in a NiagaraStation in its NiagaraNetwork, which in turn has mapped those Niagara virtuals in a lower-tiered NiagaraStation in its NiagaraNetwork, and so on.
- "Niagara virtuals to other virtual gateways" are also supported. Currently this applies to the BACnet driver, where each BacnetDevice has its own "Virtual" gateway that provides access to virtual components that represent BACnet objects. Now you can map those Bacnet virtual components as Niagara virtual components in that NiagaraStation. This saves having to create Bacnet proxy points in JACE stations for Px access at the Supervisor.
- "Shorter virtual *ord* slot paths" now result, instead of the longer virtual ord syntax previously used (which included encoded data on the target slot's facets and config flags). Shorter slot paths were necessary for the "virtuals to virtuals" feature. They also allow easy editing of slot paths, if necessary. Simplified ords are facilitated by a "Virtual Cache" mechanism, which stores the extra data removed from virtual slot paths. This virtual cache operates transparently, yet has a component interface and Workbench view for advanced users who want to inspect or remove cached items. For related details, see "Ords for Niagara virtual components" on page 2-42 and "Niagara virtuals cache (Virtual Policies)" on page 2-42.

The following sections provide more details about Niagara virtual components:

- Licensing and application overview
- About the Niagara Virtual Device Ext
- Security and Niagara virtuals
- Views on Niagara virtuals
- Actions on Niagara virtuals
- Niagara virtuals in Px views
- Spy page diagnostics for Niagara virtual components
- Localization support for Niagara virtuals

Licensing and application overview

Details on licensing, advantages, and limitations of Niagara virtual components are in these sections:

- Licensing
- Advantages
- Limitations

Niagara virtual component licensing

Virtual components under a NiagaraNetwork are a standard licensed feature of any Supervisor. In the Supervisor host's license, the boolean "`virtual`" attribute in the niagaraDriver feature line enables this:

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

This option *may not* be licensed in most JACE hosts, nor needs to be—unless a JACE's station is the "middle tier" of a "Niagara virtual to Niagara virtual" mapping back to an AX-3.7 or later Supervisor. In this case, the JACE's niagaraDriver feature requires the "`virtual`" option, and it must run AX-3.7 or later.

Note an AX-3.7 Supervisor can effectively access "Niagara virtuals of *Bacnet virtuals*" in a AX-3.7 JACE station, without the JACE requiring its niagaraDriver license feature to have this "`virtual`" option.

An AX-3.7 Supervisor can also access Niagara virtuals of most components in AX-3.6 and AX-3.5 stations; however, no "Niagara virtuals of virtuals" (Bacnet virtuals or Niagara virtuals) are possible.

Niagara virtual component advantages

Use of Niagara virtuals in a Supervisor provides the same "on demand subscription" of remote realtime data on Px pages as does proxy points, including right-click popup menus to issues actions. However, the Supervisor station overhead from scores of persisted Niagara proxy points is gone—instead the only persisted items are simply the ORDs to virtuals within Px widgets. This results in fewer resources consumed, along with faster startup of the Supervisor station

Additionally, Niagara virtuals provide a possible savings in engineering time—as point "discovery" and creation using the Bql Query Builder is *not needed*. Instead, Niagara virtual components expand under a NiagaraStation's "Virtual" gateway to reflect the *entire component tree* of that station. So, you can simply drag and drop virtuals onto Px pages and select appropriate settings in the popup "Make Widget" dialogs.

Further, *writable* Niagara virtuals provide user access to almost any property in a PxPage graphic, for both display and *adjustment* purposes. For example, to expose the high alarm limit for point, you simply expand its alarm extension to reveal child containers, drag the "Offnormal Algorithm" virtual onto the Px page, and in the Make Widget popup dialog, select the ORD to the "High Limit" property. Any beforehand creation of a Niagara proxy point for this is not needed.

For related details, see "Niagara virtuals in Px views" on page 2-47.

Niagara virtual component limitations

Because Niagara virtual components are not persisted in the Supervisor's station database, *links to and from Niagara virtuals are not supported*. Therefore, values from remote stations needed within station logic require Niagara proxy points. A simple example is a Math "Average" object that averages zone temperatures originating in different JACE stations. Each input to the Math object would require a Niagara proxy point of the relevant (remote station) data source.

Point extensions under Niagara virtuals are also not supported, for example alarm and history extensions—although from a "best practices" perspective, such extensions are often misplaced in Niagara proxy points. For further details on this, see "Best practices for Niagara proxy points" on page 2-25.

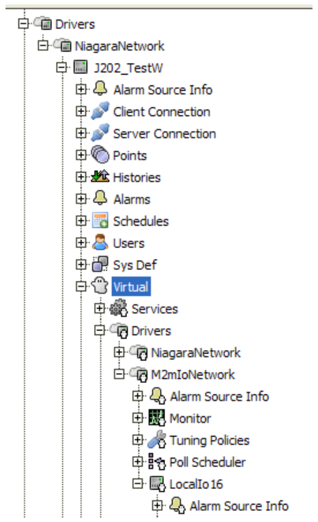
For related details, see "About the Niagara Virtual Device Ext" on page 2-40 and "Views on Niagara virtuals" on page 2-46.

About the Niagara Virtual Device Ext

Note: *Prior to AX-3.7, this component was the "NiagaraVirtualGateway", with all child virtual components sourced from the* `niagaraDriver` *module. Starting in AX-3.7, the virtual gateway architecture changed such that a separate* `niagaraVirtual` *module is used. Note that Niagara virtuals operation is essentially unchanged, except for improvements. For related details, see "Niagara virtuals in AX-3.7" on page 2-38.*

Every NiagaraStation component in a station's NiagaraNetwork has a frozen NiagaraVirtualDeviceExt (**`Virtual`**) slot, at the same level as other device extension slots (Points, Schedules, etc.).

Figure 2-45 *NiagaraVirtualDeviceExt expanded in Supervisor station*



Successfully accessing components under this Virtual device extension (gateway) dynamically adds them as Niagara virtual components (or Niagara "virtuals") while they are subscribed, but they exist only in memory—they are not persisted in the station database like proxy points.

The following sections provide more details on a Niagara Virtual Device Ext:

- Niagara Virtual gateway requirements
- Gateway memory cache and Refresh action
- Ords for Niagara virtual components

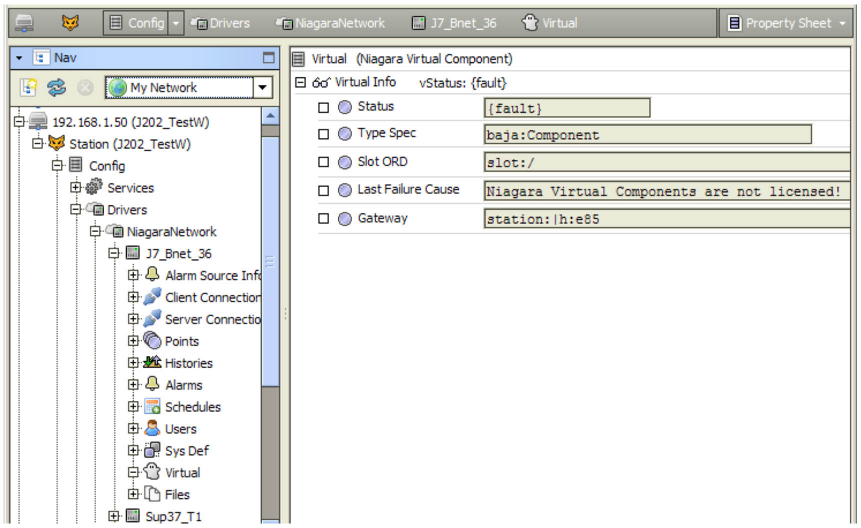
Niagara Virtual gateway requirements

Note: *Any Virtual gateway provides virtual component access only if the following are all true:*

- The local station's host is so licensed. Typically, licensing applies mainly to a Supervisor host—for further details see "Niagara virtual component licensing" on page 2-39.
  Therefore, in most JACE stations, any NiagaraStation's Virtual gateway is non-functional.
- 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—see "Security and Niagara virtuals" on page 2-45 for more details.

The property sheet of a Niagara Virtual device extension (gateway) has a "Virtual Info" slot that provides a number of status properties, including a "Last Failure Cause" property, as shown in Figure 2-46.

Figure 2-46 *Virtual Info properties of Niagara Virtual Gateway*



Virtual (Niagara Virtual Component)
Virtual Info vStatus: {fault}

Status	{fault}
Type Spec	baja:Component
Slot ORD	slot:/
Last Failure Cause	Niagara Virtual Components are not licensed!
Gateway	station:|h:e85

In the Figure 2-46 example, the Virtual gateway has a fault status, because even though the station's "Virtuals Enabled" property is true, the (JACE) host running this station is not licensed for Niagara virtuals. In this typical case, note the last failure cause is "Niagara Virtual Components are not licensed".

In a Supervisor station, this specific failure cause would not appear. However, a Niagara Virtual gateway in a Supervisor station may still be disabled, or even in fault (say, if the remote station is running an earlier release than AX-3.4). In any case, the failure cause property provides fault details.

Gateway memory cache and Refresh action

As a Baja virtual gateway, the Niagara Virtual device extension (gateway) only loads what is needed or "activated" as virtual components in the memory space of the station. For a general overview of how this works, see "Gateway activation" on page 1-24.

Activation of a Niagara Virtual gateway results in some number of "client virtual subscriptions" to those components exposed, when any of the following occurs:

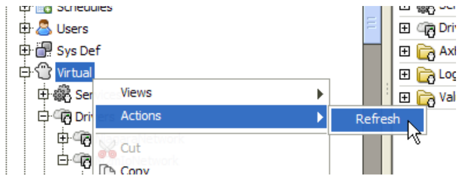
- a Virtual device extension is expanded in the Workbench Nav tree
- a Virtual device extension is expanded in its Workbench property sheet
- a Px page (with widgets bound to virtuals under that Virtual device extension) is being viewed

By default, viewing activated components places them in a memory "cache" for some minimum number of seconds, where they remain as long as being active (viewed). After some maximum time of inactivity, virtual components may be automatically deleted from this cache. By default, inactive cache life is around 1 minute. In most cases default settings are fine, however, associated cache settings for Niagara virtual components can be adjusted by changing entries in the host's (Supervisor's) `!\lib\system.properties` file. Note this file includes self-doc comments about these settings.

Starting in AX-3.7, the cache of frequently-accessed virtual components is also stored *persistently* in one or more *files* under the station's folder, by default. These "niagara virtual archive" (`.nva`) files are under the station's default `niagaraDriver_nVirtual` subfolder, as `cache1.nva`, `cache2.nva`, and so on. They are typically created upon station shutdown, and provide faster Px page loading. For related details, see "Niagara virtuals cache (Virtual Policies)" on page 2-42.

Refresh action Note that if engineering a job while configuration changes are still being made in a remote station (modeled by a NiagaraStation in the local NiagaraNetwork), it may be necessary to use the right-click **`Refresh`** action on the Virtual device extension, as shown being done in Figure 2-47.

Figure 2-47 Refresh action on Niagara Virtual gateway



As a general rule, a Refresh action is typically needed when the Nav tree contents for Niagara virtuals under a station are “not as expected.”

Ords for Niagara virtual components

Starting in AX-3.7, Niagara virtual components use a “simplified”, shorter ord syntax, where an example ord may look similar to:

```
station:|slot:/Drivers/NiagaraNetwork/J202_Test/virtual|virtual:/Logic/
RTUsAvgRA/out
```

Formerly, ords for Niagara virtuals included encoded data about facets and config flags of the target slot. Now this “extra information” is stored in a “virtual cache” of the parent NiagaraNetwork’s “Virtual Policies” container. For related details, see “Niagara virtuals cache (Virtual Policies)” on page 2-42.

Note that because “virtual of virtuals” are now supported, the ord for a Niagara virtual may include the slot path of two or more virtual gateways. For example:

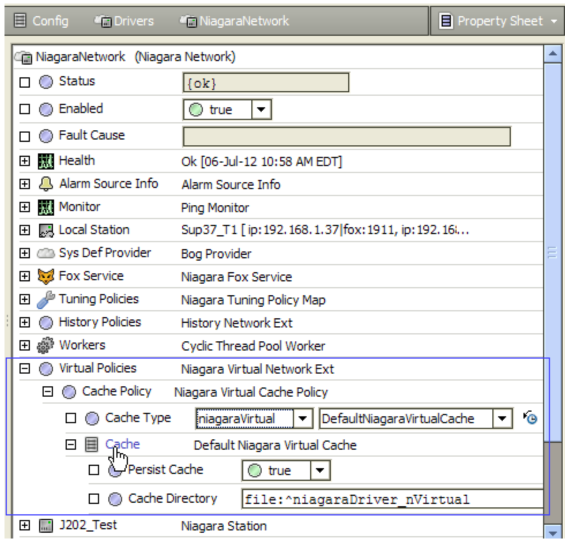
```
station:|slot:/Drivers/NiagaraNetwork/J7_Bnet_36/virtual|virtual:/Drivers/
BacnetNetwork/J4_R2c_99/virtual/virtual/binaryInput_3/
presentValue|slot:value
```

includes a slot path through the “Bacnet virtual gateway” of BacnetDevice “J4_R2c_99” to the Present_Value property of a BACnet Binary Input object.

Niagara virtuals cache (Virtual Policies)

Starting in AX-3.7, the NiagaraNetwork in any station has a “Virtual Policies” container slot (Figure 2-54).

Figure 2-48 Example Virtual Policies container expanded in Supervisor station to show “Cache” child



NiagaraNetwork (Niagara Network)

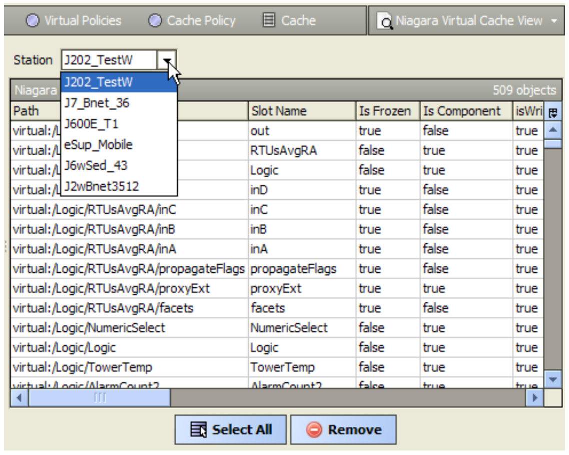
Status	{ok}
Enabled	true
Fault Cause	
Health	Ok [06-Jul-12 10:58 AM EDT]
Alarm Source Info	Alarm Source Info
Monitor	Ping Monitor
Local Station	Sup37_T1 [ip:192.168.1.37|fox:1911, ip:192.16...
Sys Def Provider	Bog Provider
Fox Service	Niagara Fox Service
Tuning Policies	Niagara Tuning Policy Map
History Policies	History Network Ext
Workers	Cyclic Thread Pool Worker
Virtual Policies	Niagara Virtual Network Ext
Cache Policy	Niagara Virtual Cache Policy
Cache Type	niagaraVirtual DefaultNiagaraVirtualCache
Cache	Default Niagara Virtual Cache
Persist Cache	true
Cache Directory	file:^niagaraDriver_nVirtual
J202_Test	Niagara Station

For a Supervisor’s NiagaraNetwork, or for any station’s NiagaraNetwork that uses Niagara virtuals, this provides access to cached data on virtual components. For any other station (typically most JACE stations), the Virtual Policies container is unused.

From the NiagaraNetwork’s property sheet, expand this container to see the available configuration properties and contained “Cache”. Typically, default property values are appropriate—for more details see “Virtual Policies (Cache) properties” on page 2-44.

Double-click the **`Cache`** component for its default **`Niagara Virtual Cache View`**.

Figure 2-49 *Niagara Virtual Cache View of NiagaraNetwork's cache (with station select drop-down)*



Station: J202_TestW

J202_TestW
J7_Bnet_36
J600E_T1
eSup_Mobile
J6wSed_43
J2wBnet3512

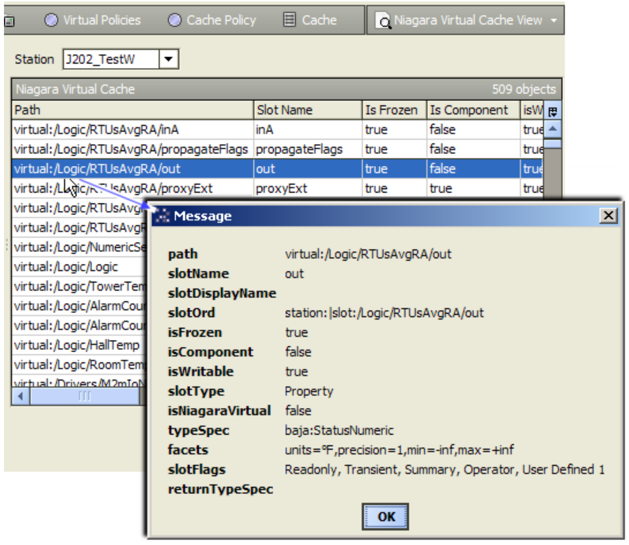
Niagara — 509 objects

Path	Slot Name	Is Frozen	Is Component	isWri
virtual:/L	out	true	false	true
virtual:/L	RTUsAvgRA	false	true	true
virtual:/L	Logic	false	true	true
virtual:/L	inD	true	false	true
virtual:/Logic/RTUsAvgRA/inC	inC	true	false	true
virtual:/Logic/RTUsAvgRA/inB	inB	true	false	true
virtual:/Logic/RTUsAvgRA/inA	inA	true	false	true
virtual:/Logic/RTUsAvgRA/propagateFlags	propagateFlags	true	false	true
virtual:/Logic/RTUsAvgRA/proxyExt	proxyExt	true	true	true
virtual:/Logic/RTUsAvgRA/facets	facets	true	false	true
virtual:/Logic/NumericSelect	NumericSelect	false	true	true
virtual:/Logic/Logic	Logic	false	true	true
virtual:/Logic/TowerTemp	TowerTemp	false	true	true
virtual:/Logic/AlarmCount2	AlarmCount2	false	true	true

As shown in Figure 2-49, this tabular view lists cached virtual ords, with a "station select" control 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 dialog showing details on data in the virtual cache (Figure 2-50).

Figure 2-50 *Double-click row in Niagara Virtual Cache View for cached details on virtual component*



Station: J202_TestW

Niagara Virtual Cache — 509 objects

Path	Slot Name	Is Frozen	Is Component	isW
virtual:/Logic/RTUsAvgRA/inA	inA	true	false	true
virtual:/Logic/RTUsAvgRA/propagateFlags	propagateFlags	true	false	true
virtual:/Logic/RTUsAvgRA/out	out	true	false	true
virtual:/L…sAvgRA/proxyExt	proxyExt	true	true	true
virtual:/Logic/RTUsAvg				
virtual:/Logic/RTUsAvgR				
virtual:/Logic/NumericSe				
virtual:/Logic/Logic				
virtual:/Logic/TowerTem				
virtual:/Logic/AlarmCou				
virtual:/Logic/AlarmCou				
virtual:/Logic/HallTemp				
virtual:/Logic/RoomTem				
virtual:/Drivers/M2mIoN				

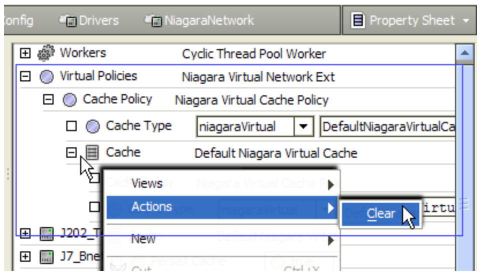
path	virtual:/Logic/RTUsAvgRA/out
slotName	out
slotDisplayName	
slotOrd	station:|slot:/Logic/RTUsAvgRA/out
isFrozen	true
isComponent	false
isWritable	true
slotType	Property
isNiagaraVirtual	false
typeSpec	baja:StatusNumeric
facets	units=°F,precision=1,min=-inf,max=+inf
slotFlags	Readonly, Transient, Summary, Operator, User Defined 1
returnTypeSpec	

As shown, 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's property sheet (Figure 2-51).

Figure 2-51 *Clear action on Virtual Policies > Cache component, from NiagaraNetwork property sheet*



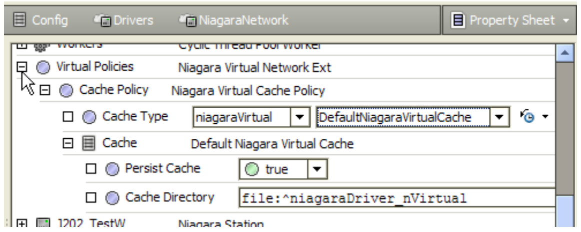
Workers	Cyclic Thread Pool Worker
Virtual Policies	Niagara Virtual Network Ext
Cache Policy	Niagara Virtual Cache Policy
Cache Type	niagaraVirtual DefaultNiagaraVirtualCa
Cache	Default Niagara Virtual Cache
J202_T	
J7_Bne	

Clear removes cached data for virtual components across *all NiagaraStations* in the NiagaraNetwork.

Virtual Policies (Cache) properties

Note: *In most cases it is recommended you leave configuration properties at defaults, as shown in Figure 2-52.*

Figure 2-52 *Virtual Policies expanded in NiagaraNetwork property sheet*



Config | Drivers | NiagaraNetwork | Property Sheet

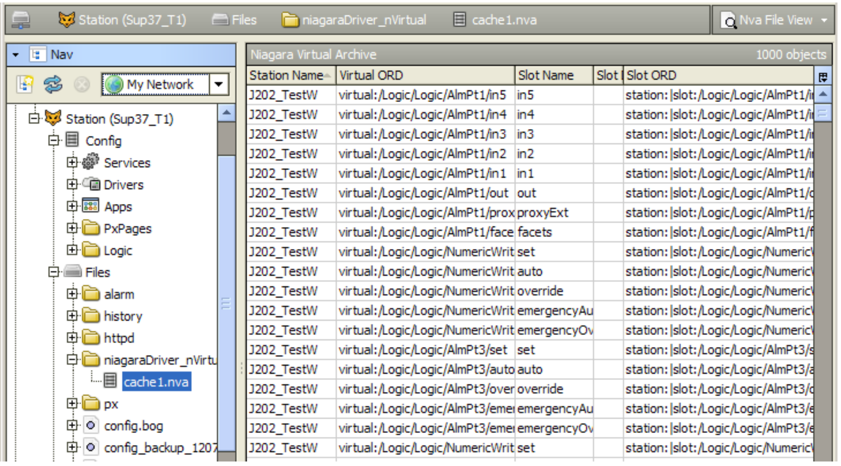
Workers	Cyclic Thread Pool Worker
Virtual Policies	Niagara Virtual Network Ext
Cache Policy	Niagara Virtual Cache Policy
Cache Type	niagaraVirtual / DefaultNiagaraVirtualCache
Cache	Default Niagara Virtual Cache
Persist Cache	true
Cache Directory	file:^niagaraDriver_nVirtual
J202_TestW	Niagara Station

Properties and slots of the Niagara Virtual Cache Policy are as follows:

- **Cache Type**
  Cache type has two fields as drop-down choices, with currently the first fixed at **`niagaraVirtual`**, and the second as either:
  - **`DefaultNiagaraVirtualCache`** — (default), so that ords and associated data from accessing virtual components is cached in memory. Also see "Gateway memory cache and Refresh action" on page 2-41.
  - **`NullCache`** — Nothing is cached. In this case, virtual component access results in more network traffic and graphics will be slower to load.
- **Cache**
  Cache is the component representation of virtual cache data, with the default "Niagara Virtual Cache View" (see Figure 2-49 on page 43). Configuration properties include:
  - **`Persist Cache`** — (default is true) where some slot data for virtual components is stored persistently in `cache`*`N`*`.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.
  - **`Cache Directory`** — The file ord to the folder in which `cache`*`N`*`.nva` file(s) are created when "Persist Cache" is true. The default ord is `file:^niagaraDriver_nVirtual`

Note persisted virtual cache files can also be examined in Workbench by double-clicking for a "Nva File View", as shown in Figure 2-53.

Figure 2-53 *Workbench provides a default "Nva File View" for persisted Niagara virtual cache files.*



Station (Sup37_T1) | Files | niagaraDriver_nVirtual | cache1.nva | Nva File View

Nav — My Network
Station (Sup37_T1)
Config
Services
Drivers
Apps
PxPages
Logic
Files
alarm
history
httpd
niagaraDriver_nVirtu
cache1.nva
px
config.bog
config_backup_1207

Niagara Virtual Archive — 1000 objects

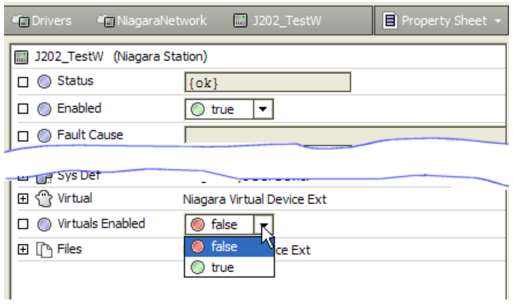
Station Name	Virtual ORD	Slot Name	Slot I	Slot ORD
J202_TestW	virtual:/Logic/Logic/AlmPt1/in5	in5		station:|slot:/Logic/Logic/AlmPt1/i
J202_TestW	virtual:/Logic/Logic/AlmPt1/in4	in4		station:|slot:/Logic/Logic/AlmPt1/i
J202_TestW	virtual:/Logic/Logic/AlmPt1/in3	in3		station:|slot:/Logic/Logic/AlmPt1/i
J202_TestW	virtual:/Logic/Logic/AlmPt1/in2	in2		station:|slot:/Logic/Logic/AlmPt1/i
J202_TestW	virtual:/Logic/Logic/AlmPt1/in1	in1		station:|slot:/Logic/Logic/AlmPt1/i
J202_TestW	virtual:/Logic/Logic/AlmPt1/out	out		station:|slot:/Logic/Logic/AlmPt1/o
J202_TestW	virtual:/Logic/Logic/AlmPt1/prox	proxyExt		station:|slot:/Logic/Logic/AlmPt1/p
J202_TestW	virtual:/Logic/Logic/AlmPt1/face	facets		station:|slot:/Logic/Logic/AlmPt1/f
J202_TestW	virtual:/Logic/Logic/NumericWrit	set		station:|slot:/Logic/Logic/Numeric
J202_TestW	virtual:/Logic/Logic/NumericWrit	auto		station:|slot:/Logic/Logic/Numeric
J202_TestW	virtual:/Logic/Logic/NumericWrit	override		station:|slot:/Logic/Logic/Numeric
J202_TestW	virtual:/Logic/Logic/NumericWrit	emergencyAu		station:|slot:/Logic/Logic/Numeric
J202_TestW	virtual:/Logic/Logic/NumericWrit	emergencyOv		station:|slot:/Logic/Logic/Numeric
J202_TestW	virtual:/Logic/Logic/AlmPt3/set	set		station:|slot:/Logic/Logic/AlmPt3/s
J202_TestW	virtual:/Logic/Logic/AlmPt3/auto	auto		station:|slot:/Logic/Logic/AlmPt3/a
J202_TestW	virtual:/Logic/Logic/AlmPt3/over	override		station:|slot:/Logic/Logic/AlmPt3/o
J202_TestW	virtual:/Logic/Logic/AlmPt3/emer	emergencyAu		station:|slot:/Logic/Logic/AlmPt3/e
J202_TestW	virtual:/Logic/Logic/AlmPt3/emer	emergencyOv		station:|slot:/Logic/Logic/AlmPt3/e
J202_TestW	virtual:/Logic/Logic/NumericWrit	set		station:|slot:/Logic/Logic/Numeric

In the example station shown above there is only one virtual cache file. However, a large Supervisor station will typically have multiple virtual cache files.

Security and Niagara virtuals

Along with a Virtual device extension, each NiagaraStation has an associated Boolean property, “Virtuals Enabled”, found near the bottom of any NiagaraStation’s property sheet (Figure 2-54).

Figure 2-54 *“Virtuals Enabled” is false by default*



J202_TestW (Niagara Station)	
Status	{ok}
Enabled	true
Fault Cause	
Sys Def	
Virtual	Niagara Virtual Device Ext
Virtuals Enabled	false
Files	ce Ext

This property must be true before Niagara virtual components for this station can be accessed. However, it defaults to false as a security measure, as when initially enabled, *all components* in the remote station will be accessible to *any user* with permissions on this NiagaraStation’s Virtual gateway.

For example, any user with admin-level permissions on the gateway will be able to view all Users in the remote station’s UserService, simply by expanding the property sheet of the virtual component for the UserService. Or, such a user will be able to expand the station’s PlatformServices virtual component, and have access to critical properties or child services (including an action to restart station/reboot host)!

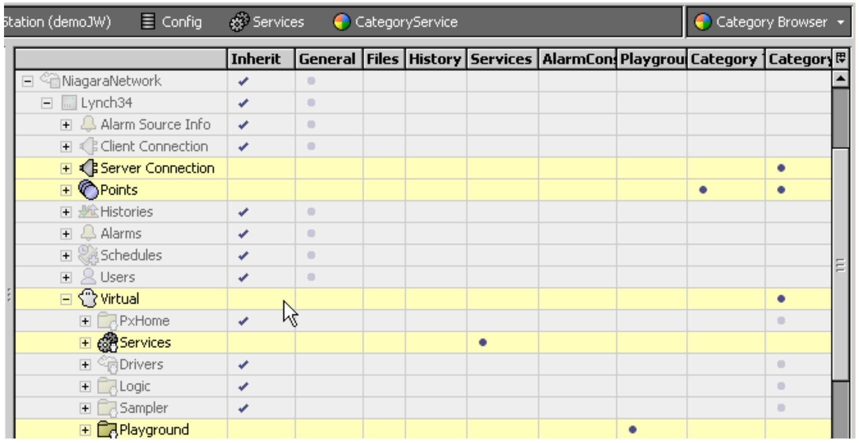
Therefore, *before* enabling any Niagara Virtual gateway, it is recommended to restrict user access to it, (assign it to a restricted category). Then, on Niagara virtual components *under* that gateway, assign appropriate category(ies)—before re-assigning less restrictive category(ies) to the parent gateway.

Note: *Persisted category assignments for individual virtual components is an important feature. Originally, only the persisted parent virtual gateway component could be assigned to one or more categories—with all child virtual components inheriting those categories. Now with this feature, categories assigned to active virtual components are safely maintained even after virtuals are removed from the station’s memory cache.*

Assign categories to Niagara virtual components using “normal component” methods, that is either by

- with the Virtual gateway expanded, right-click any child virtual to go to its Category Sheet view, or:
- using the Category Browser view of the local station’s CategoryService, expanding the Niagara Virtual gateway under any NiagaraStation in the NiagaraNetwork (see Figure 2-55).

Figure 2-55 *Assigning categories to Niagara virtual components using the Category Browser*



	Inherit	General	Files	History	Services	AlarmCon:	Playgrou	Category	Category
NiagaraNetwork	✓	●							
Lynch34	✓	●							
Alarm Source Info	✓	●							
Client Connection	✓	●							
Server Connection									●
Points								●	●
Histories	✓	●							
Alarms	✓	●							
Schedules	✓	●							
Users	✓	●							
Virtual									●
PxHome	✓								●
Services					●				
Drivers	✓								●
Logic	✓								●
Sampler	✓								●
Playground							●		

Be careful to **`Save`** after making any needed category changes to the Niagara Virtual gateway or virtual components. For more information about station security and categories, refer to the sections “About Security” and “CategoryService” in the *NiagaraAX User Guide*.

Views on Niagara virtuals

Niagara virtual components provide a *subset* of the views as normal components, where the default **`Property Sheet`** view and **`Category Sheet`** view (for security) provide the most utility—see "Security and Niagara virtuals" on page 2-45. Special "manager" views on virtuals are not available.

Because of the transient (non-persisted) nature of virtuals, the other common types of views on Niagara virtuals can be summarized as follows:

- Wire Sheet — *not available*, as links to/from any virtual components are not supported.
- Slot Sheet — available, but little practical application—as any changes made to slots (config flags, new slot, etc) are not persisted.
- Link Sheet — *not available*, as links to/from any virtual components are not supported.

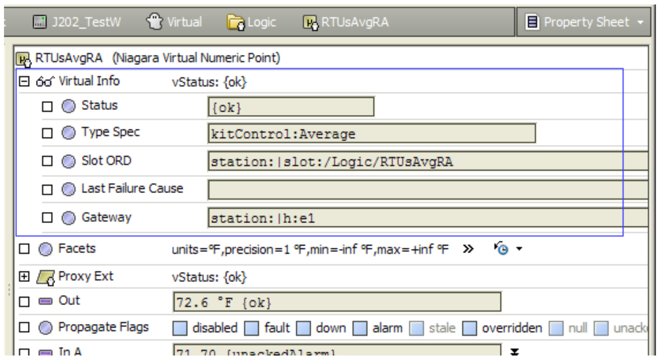
Note: *Creating a Px view directly on a Niagara virtual or the Niagara Virtual gateway (using "New View" from the right-click Workbench menu), is not supported/available. However, you can create such Px views on other persisted components in the station, for example on Folder or IconFolder components, and then add Px widgets with Niagara virtual component bindings. See "Niagara virtuals in Px views" on page 2-47.*

Property sheet access of any Niagara virtual provides a "Virtual Info" slot at the top of the view, see the section "About Virtual Info".

About Virtual Info

In addition to the "Virtual Info" properties available on a Niagara Virtual device extension, each child Niagara virtual has a "Virtual Info" container with five read-only status properties (Figure 2-56).

Figure 2-56 *Virtual Info properties on any Niagara virtual component*



RTUsAvgRA (Niagara Virtual Numeric Point)

Virtual Info	vStatus: {ok}
Status	{ok}
Type Spec	kitControl:Average
Slot ORD	station:|slot:/Logic/RTUsAvgRA
Last Failure Cause	
Gateway	station:|h:e1
Facets	units=°F,precision=1 °F,min=-inf °F,max=+inf °F
Proxy Ext	vStatus: {ok}
Out	72.6 °F {ok}
Propagate Flags	☐ disabled ☐ fault ☐ down ☐ alarm ☐ stale ☐ overridden ☐ null ☐ unack

Virtual Info properties are in *addition* to the properties of the target source component, and include:

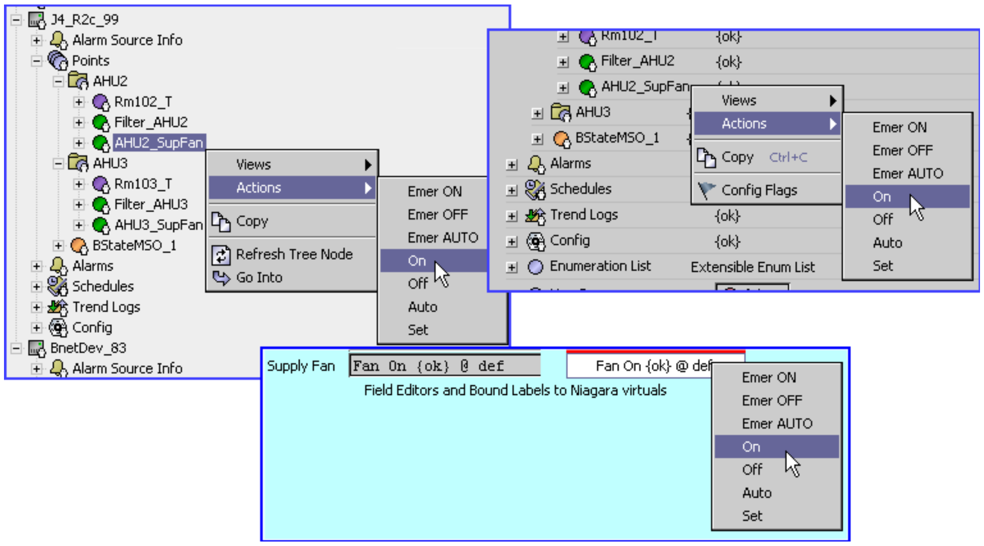
- **Virtual Status**
  For any Niagara virtual component, this is status of that virtual—not to be confused with whatever status the source (remote station) component may currently have. This status always appears on property sheets with a "`vStatus`" descriptor.
  For example, a virtual for a proxy point may have an "ok" virtual status, yet show an "Out" with a status of "down". Virtual status may initially show with a status of "stale," before changing to "ok".
- **Type Specification**
  Reflects the *`moduleName:componentType`* used for representing the Niagara virtual, for example in the property sheet view for that component.
- **Slot ORD**
  The complete slot path to the remote source component (relative to that station's slot hierarchy), where the leading "root" portion is `station:|slot:`
- **Last Failure Cause**
  The last unsuccessful attempt (if any) to activate this virtual component is explained in this text string property. May be blank if no previous unsuccessful attempt has occurred.
  Note: *Where a "Niagara virtual to virtual" path is used, a past unsuccessful attempt may include information about the comm error, for example:*
  ```
  Station Err: (J7_Bnet_36 -> J202_TestW) {down}
  ```
- **Gateway**
  The "handle" ord of the virtual gateway within the station's virtual component space.

Note that additional "spy" page information for Niagara virtuals is also available for any station. For more details, see "Spy page diagnostics for Niagara virtual components" on page 2-49."

Actions on Niagara virtuals

If a station user has invoke permissions on Niagara virtual components, they provide the same right-click actions as if interfacing directly with the source component. As shown in Figure 2-57, this applies whether from the Workbench Nav tree, property sheet, or a Px page with binding to that virtual.

Figure 2-57 *Right click actions of source component available through Niagara virtual*

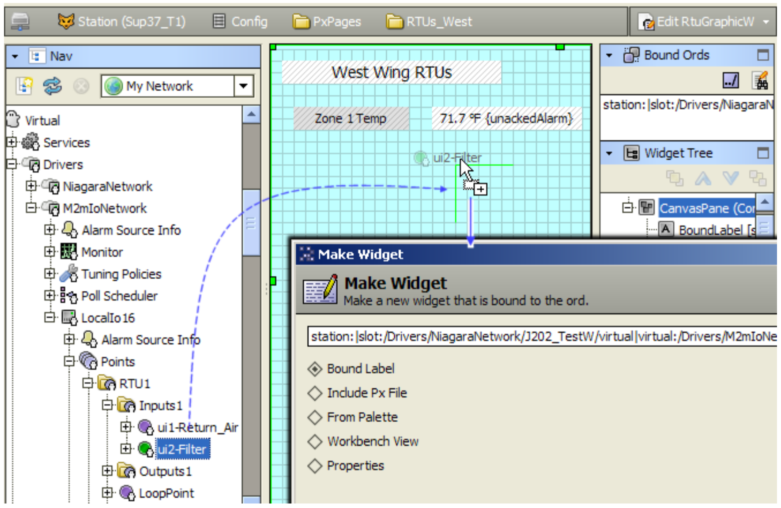


Consider this when assigning categories to Niagara virtuals. Users with invoke (I) permissions (at operator or admin levels) on those categories will have access to those associated actions. See "Security and Niagara virtuals" on page 2-45 for related information.

Niagara virtuals in Px views

The primary application of Niagara virtual components is for building Px views. Workbench allows you to "drag and drop" Niagara virtuals onto Px pages. Figure 2-58 shows a "drag and drop" of a Niagara virtual for a control point, showing the top portion of the `Make Widget` popup dialog for a bound Label.

Figure 2-58 *Drag and drop of Niagara virtual onto Px page produces Make Widget dialog*

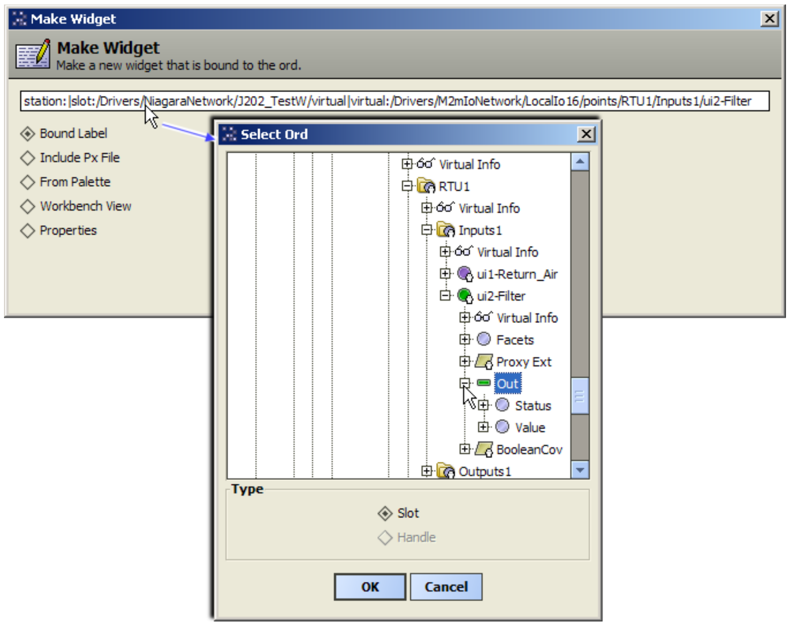


Px binding notes for Niagara virtuals

A properly added bound label (to a Niagara virtual) can provide the same Px access as if bound to Niagara proxy point. Other Px widget selections, such as a field editor (property selection) are also supported.

In general, it is recommended that you make Px bindings to a *specific property* of a Niagara virtual, rather than a whole virtual component. This can improve both performance and graphical rendering (for example, if a Bound Label, this technique can also allow the "Status" (Color) option to work).

Figure 2-59 *Double-click ORD field at top of Make Widget dialog if making Bound Label, to select property*



In Figure 2-59, note the selection of a *specific property* of the Niagara virtual, via the **`Select Ord`** popup dialog. In this example, the selected component is expanded, and the property “Out” is selected.

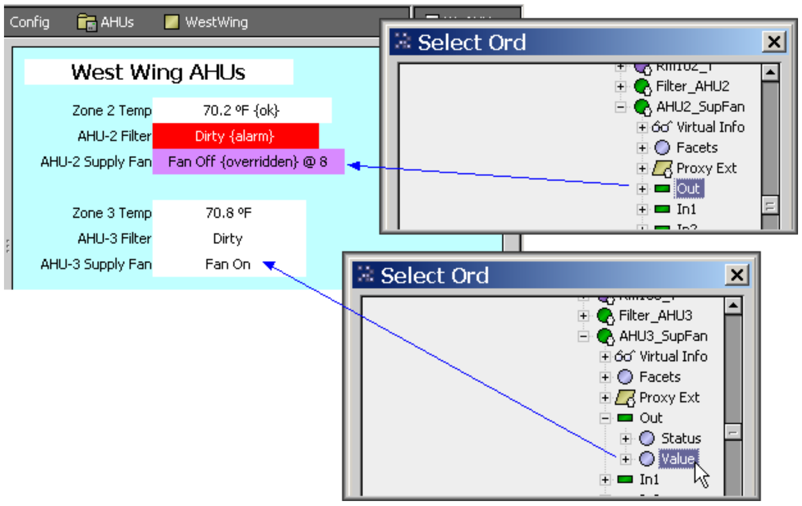
To do this, *double-click in the top field* (Ord) of the **`Make Widget`** dialog after dropping the Niagara virtual. This produces the **`Select Ord`** dialog, where you then expand the selected virtual to select its desired property (as shown being done in the Figure 2-59 example). Double-click to select and close.

If only the value or status is wanted for display in a bound label, you could further expand the Out property in the Select Ord dialog, and select either Value or Status.

Note: *You must further select “Value” or “Status” of a selected “Out” property in the case of a selected “virtual to virtual”. Otherwise, the Px value displayed is likely to be:* `vStatus {ok}` *(instead of the end target value). Alternatively, you can edit the BoundLabel Px widget’s BFormat (text) property value from the default:* `%.%` *to instead:* `%value%` *or* `%status%` *or* `%value% %status%`.

For example, consider the Figure 2-60 Px example of two groups of 3 bound labels. each bound to a Niagara virtual representing a control point.

Figure 2-60 *Example Bound Labels on Px page using different property depth*



- In the *top* group, as each bound label was added the “`Out`” property of that Niagara virtual component was selected in the **`Select Ord`** dialog.
- In the *bottom* group, as each bound label was added the “`Out`” property of that virtual component was *expanded* in the **`Select Ord`** dialog, and only the “`Value`” portion selected. Note that status information does not appear in these real-time values, and the “Status” (Color) option does not work.

In the two preceding examples, the BFormat text formatting (Format Text or "text" property), was left at a default %.% value. If needed, support for additional `text` formatting is also available Bound Labels. For example, to add leading (or trailing) static text around the *`%scripted%`* portion, after adding the widget.

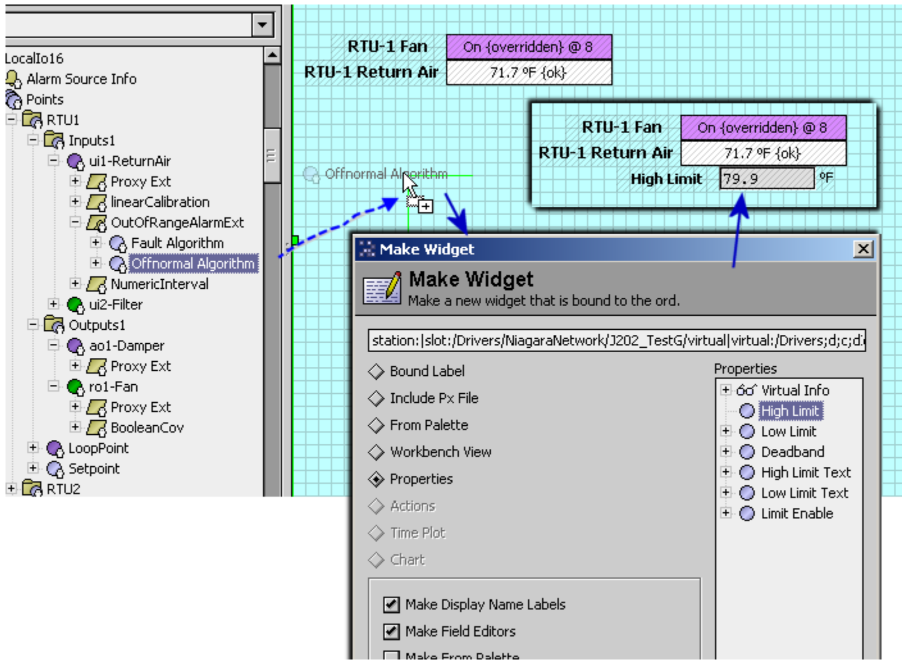
However (and again), it is more efficient to specify a particular *property* of a Niagara virtual (initially, in the **`Select Ord`** dialog before widget creation). That is, instead of adding a Px widget bound to the root of a Niagara virtual component, and then specifying a particular *`%propertyName%`* in its text formatting.

In the case of doing the latter, note that *all properties* of the target Niagara virtual become loaded from viewing the Px page, not just the single property specified by the text formatting. You can use "spy" troubleshooting pages to observe this behavior as applied to Niagara virtual components. See the next section, "Spy page diagnostics for Niagara virtual components".

Px usage of Niagara virtuals for write adjustments

Niagara virtuals have *writable* properties. This lets you expose properties that may need adjustment onto PxPages, using standard "drag and drop" techniques. Figure 2-61 shows such an example, for an alarm limit for a proxy point's alarm extension.

Figure 2-61 *Example use of writable Niagara virtual in PxPage*



If users have proper permissions on such Niagara virtuals, they can make configuration changes that are written to the (actual) source component.

Note the "class types" for Niagara virtuals include a Niagara virtual type for each of the eight NiagaraAX "control points", that is a `NiagaraVirtualNumericPoint`, `NiagaraVirtualNumericWritable`, `NiagaraVirtualBooleanPoint`, and so on. This results in better overall handling of point status in Niagara virtuals on PxPages. This also helped in development of the "export tags" feature for Supervisor auto-configuration, first introduced in AX-3.5. Starting in AX-3.7, these eight class types are sourced from the `niagaraVirtual` module, along with most other Niagara virtual component types and views.

Spy page diagnostics for Niagara virtual components

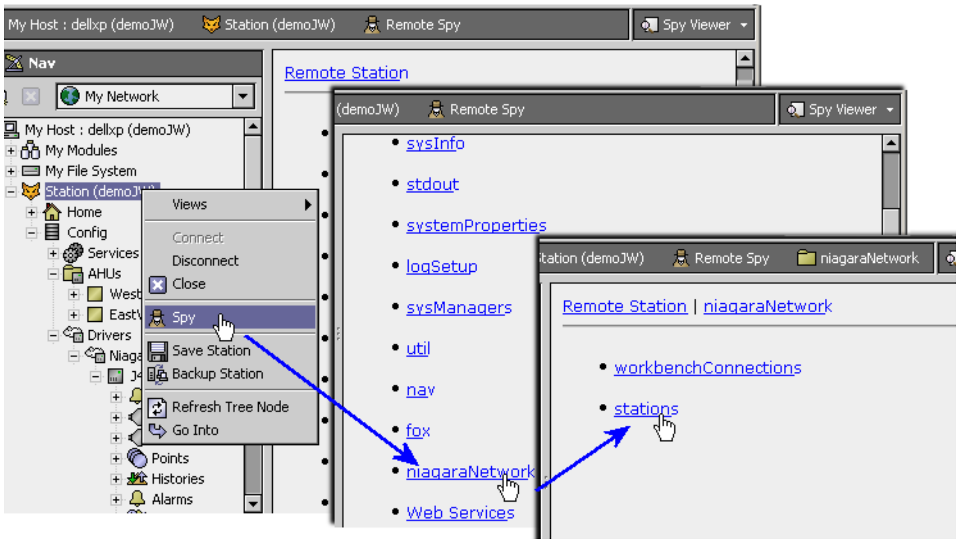
"Spy page" support was extended to help troubleshoot/diagnose usage of Niagara virtual components. Typical spy usage for Niagara virtuals is against the Supervisor station, which may include many Px views with many bindings to various Niagara virtual components.

Note: *Spy page usage is generally considered an advanced activity, and this section may not apply to all users. However, details here may help diagnose issues in Niagara virtual component setup or maintenance.*

A Supervisor station is considered a "Niagara Virtuals client" to the various remote JACE stations, which are "Niagara Virtuals servers". Therefore, in a Supervisor's spy pages for its NiagaraNetwork, you typically expect large counts for *client* Niagara Virtual connections under its various stations.

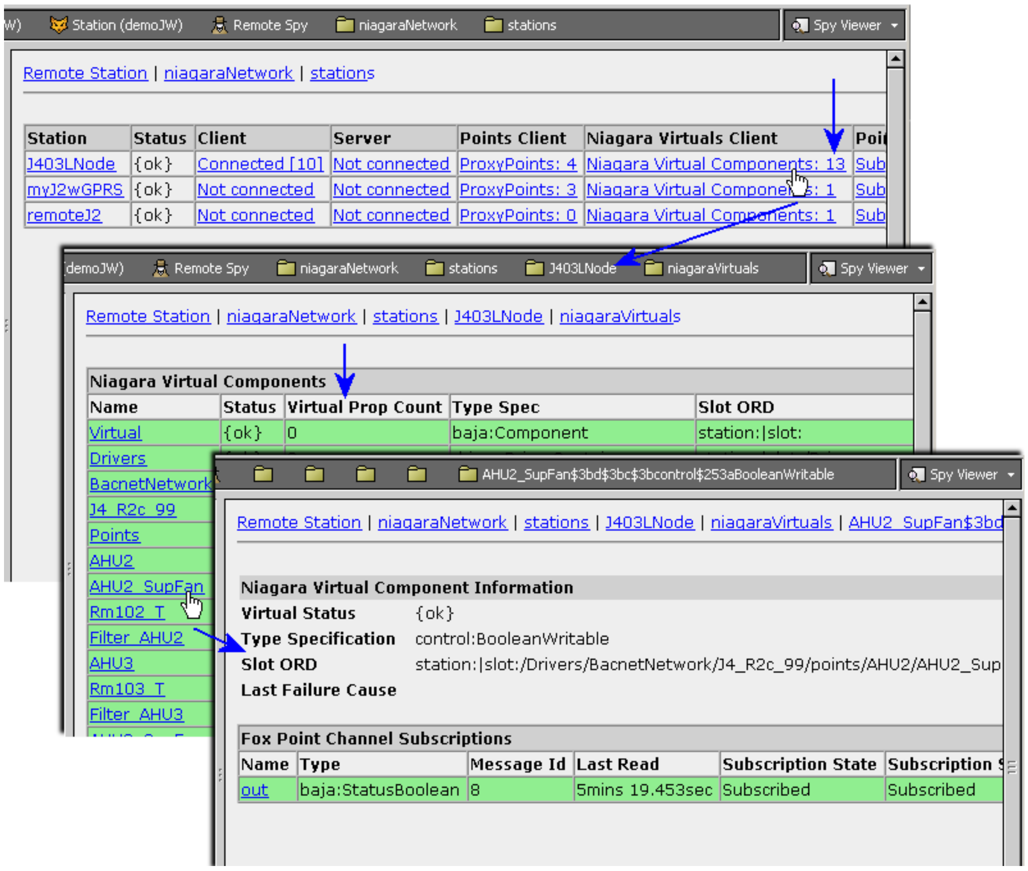
Figure 2-62 shows a spy session in Workbench launched against a open Supervisor station (right-click, select **`Spy`**, then in the **`Spy Viewer`**, **`niagaraNetwork`**, **`stations`**.

Figure 2-62 *Starting spy to look at Supervisor's usage of Niagara virtual components*



This produces a stations list, as shown in Figure 2-63.

Figure 2-63 *Continuing spy to look at Supervisor's usage of Niagara virtuals to one remote station*



Remote Station | niagaraNetwork | stations

Station	Status	Client	Server	Points Client	Niagara Virtuals Client	Poi
J403LNode	{ok}	Connected [10]	Not connected	ProxyPoints: 4	Niagara Virtual Components: 13	Sub
myJ2wGPRS	{ok}	Not connected	Not connected	ProxyPoints: 3	Niagara Virtual Components: 1	Sub
remoteJ2	{ok}	Not connected	Not connected	ProxyPoints: 0	Niagara Virtual Components: 1	Sub

Remote Station | niagaraNetwork | stations | J403LNode | niagaraVirtuals

Niagara Virtual Components

Name	Status	Virtual Prop Count	Type Spec	Slot ORD
Virtual	{ok}	0	baja:Component	station:|slot:
Drivers				
BacnetNetwork				
J4_R2c_99				
Points				
AHU2				
AHU2_SupFan				
Rm102_T				
Filter_AHU2				
AHU3				
Rm103_T				
Filter_AHU3				

Remote Station | niagaraNetwork | stations | J403LNode | niagaraVirtuals | AHU2_SupFan$3bd

Niagara Virtual Component Information

Virtual Status	{ok}
Type Specification	control:BooleanWritable
Slot ORD	station:|slot:/Drivers/BacnetNetwork/J4_R2c_99/points/AHU2/AHU2_Sup
Last Failure Cause	

Fox Point Channel Subscriptions

Name	Type	Message Id	Last Read	Subscription State	Subscription S
out	baja:StatusBoolean	8	5mins 19.453sec	Subscribed	Subscribed

As shown in Figure 2-63, in the stations list table, under the "Niagara Virtuals Client" column, you can see a count of active Niagara virtual components for each station in the Supervisor's NiagaraNetwork. If you click on a link in that count column, you see a page of the Niagara Virtual components, by name, including a "Virtual Prop Count" column that lists the number of associated properties for each one (ideally 1 property maximum for each one, for performance reasons).

Finally, as shown near the bottom of Figure 2-63, if you click a specific Niagara Virtual (by Name link), you see its "Virtual Info" properties, as well as subscription statistics on however many of its properties are currently subscribed (or attempted for subscription).

Note: *There is also a “Points/Virtual Server” column near the right side of the spy:/niagaraNetwork/stations page (at top of Figure 2-63). That column lists the count of active (remote) client subscriptions to components in the station. In a typical Supervisor station, this count will be 0 for each station in its NiagaraNetwork.*

However, you can open a spy session to any selected JACE station and observe this count, as well as subsequent links in a similar manner as shown in the preceding figures. Investigation on the “JACE” station side may also be helpful in resolving issues.

Localization support for Niagara virtuals

Localization (non-English translation support) for Niagara virtual components is provided for items that appear in the Workbench Nav tree as well as in property sheet views. This is done by using/editing lexicons for the modules `niagaraDriver` and `niagaraVirtual`, as well as for other modules in use by the Supervisor station.

Matching lexicon files should also be installed in JACE hosts running subordinate JACEs.

About Sys Def components

Every NiagaraNetwork has a number of “Sys Def” components, both at the network level and at each child NiagaraStation level. Currently, there is no direct, standard, application for these “system definition” components—they exist mainly for use by NiagaraAX developers.

Sys Def facilitates an API interface to define the organization of stations in a Niagara system in a known hierarchy, and allows synchronization of basic information “up” that hierarchy. Developer usage may be especially useful in large “enterprise class” NiagaraAX installations that use “tiered” Supervisors.

At the time of this document, Sys Def components have no special Workbench views, apart from the standard set of views (property sheet, slot sheet, and so on).

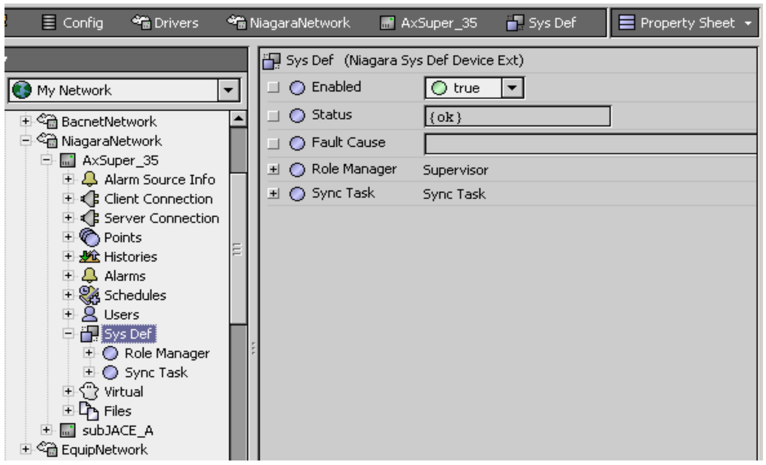
For limited basic information, see the following subsections:

- About the Niagara Sys Def Device Ext
- About network level Sys Def components

About the Niagara Sys Def Device Ext

Any **`NiagaraStation`** has a Sys Def device extension (NiagaraSysDefDeviceExt), with two child components, as shown in Figure 2-64.

Figure 2-64 *NiagaraStation’s Sys Def Device Ext property sheet*



Sys Def (Niagara Sys Def Device Ext)

Enabled	true
Status	{ok}
Fault Cause	
Role Manager	Supervisor
Sync Task	Sync Task

By default, the Sys Def device extension is Enabled.

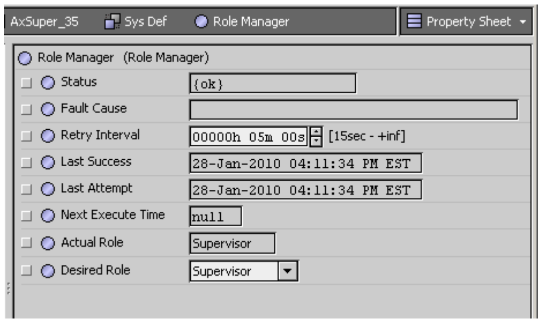
Child components of the Sys Def device extension are:

- Role Manager
- Sync Task

About the Role Manager

The Role Manager of a Niagara Sys Def Device Ext reflects two configuration properties, as well as several read-only status properties (Figure 2-65).

Figure 2-65 *Sys Def Device Ext's Role Manager property sheet*



AxSuper_35 | Sys Def | Role Manager | Property Sheet

Role Manager (Role Manager)

Status	{ok}
Fault Cause	
Retry Interval	00000h 05m 00s [15sec - +inf]
Last Success	28-Jan-2010 04:11:34 PM EST
Last Attempt	28-Jan-2010 04:11:34 PM EST
Next Execute Time	null
Actual Role	Supervisor
Desired Role	Supervisor

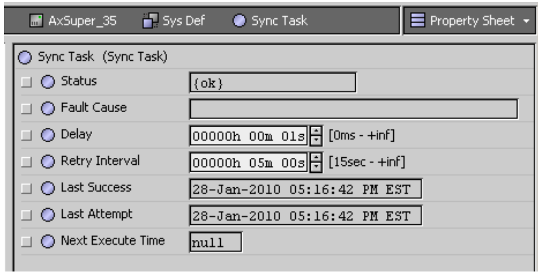
The key configuration property, **`Desired Role`**, defines the relationship of the *remote station* (represented by the parent NiagaraStation) to the local station, as either:

- Peer — (default) a "peer" relationship exists.
- Supervisor — the remote station represented by this NiagaraStation is this station's Supervisor.
- Subordinate — the remote station represented by this NiagaraStation is this station's subordinate (typically a JACE station).

About the Sync Task

The Sync Task of a Niagara Sys Def Device Ext reflects two configuration properties, as well as several read-only status properties (Figure 2-66).

Figure 2-66 *Sys Def Device Ext's Sync Task property sheet*



AxSuper_35 | Sys Def | Sync Task | Property Sheet

Sync Task (Sync Task)

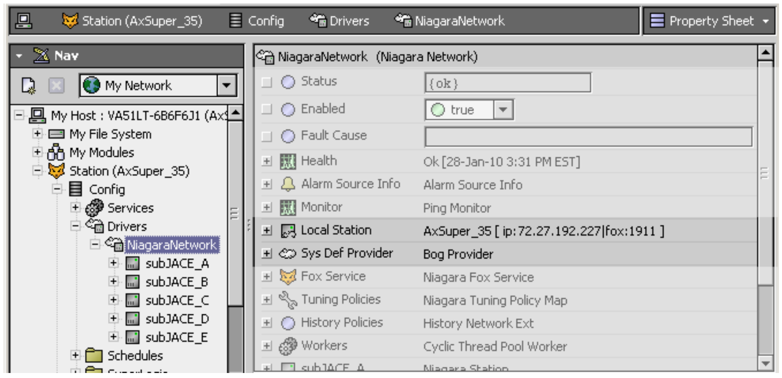
Status	{ok}
Fault Cause	
Delay	00000h 00m 01s [0ms - +inf]
Retry Interval	00000h 05m 00s [15sec - +inf]
Last Success	28-Jan-2010 05:16:42 PM EST
Last Attempt	28-Jan-2010 05:16:42 PM EST
Next Execute Time	null

This component propagates changes to Sys Def components between stations that are in a "Supervisor/subordinate" relationship.

About network level Sys Def components

The property sheet of any **`NiagaraNetwork`** has two Sys Def-related container components, as shown in Figure 2-67.

Figure 2-67 *Sys Def components in NiagaraNetwork property sheet*



NiagaraNetwork (Niagara Network)

Status	{ok}
Enabled	true
Fault Cause	
Health	Ok [28-Jan-10 3:31 PM EST]
Alarm Source Info	Alarm Source Info
Monitor	Ping Monitor
Local Station	AxSuper_35 [ip:72.27.192.227|fox:1911]
Sys Def Provider	Bog Provider
Fox Service	Niagara Fox Service
Tuning Policies	Niagara Tuning Policy Map
History Policies	History Network Ext
Workers	Cyclic Thread Pool Worker

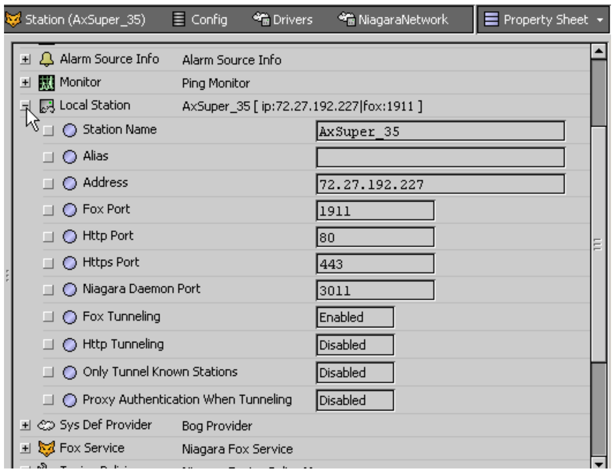
These network level components are:

- Local Station (LocalSysDefStation)
- Sys Def Provider (Bog Provider)

About the Sys Def Local Station

The Local Station reflects a collection of (read-only) "Sys Def" properties, as shown in Figure 2-68.

Figure 2-68 *Sys Def "Local Station" expanded in property sheet*



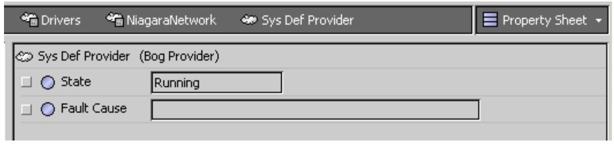
Alarm Source Info	Alarm Source Info	
Monitor	Ping Monitor	
Local Station	AxSuper_35 [ip:72.27.192.227|fox:1911]	
Station Name		AxSuper_35
Alias		
Address		72.27.192.227
Fox Port		1911
Http Port		80
Https Port		443
Niagara Daemon Port		3011
Fox Tunneling		Enabled
Http Tunneling		Disabled
Only Tunnel Known Stations		Disabled
Proxy Authentication When Tunneling		Disabled
Sys Def Provider	Bog Provider	
Fox Service	Niagara Fox Service	

These properties would be "sync'ed up" to a remote Supervisor station (if the local station was defined as its subordinate).

About the Sys Def Provider

The Sys Def Provider (BogProvider) is the API handler for queries about the "Sys Def" hierarchy, for any remote stations configured as subordinates. It persists these definitions with child "ProviderStations" in the local station's bog file (as the default storage method).

Figure 2-69 *Sys Def "Sys Def Provider" (default) expanded in NiagaraStation property sheet*

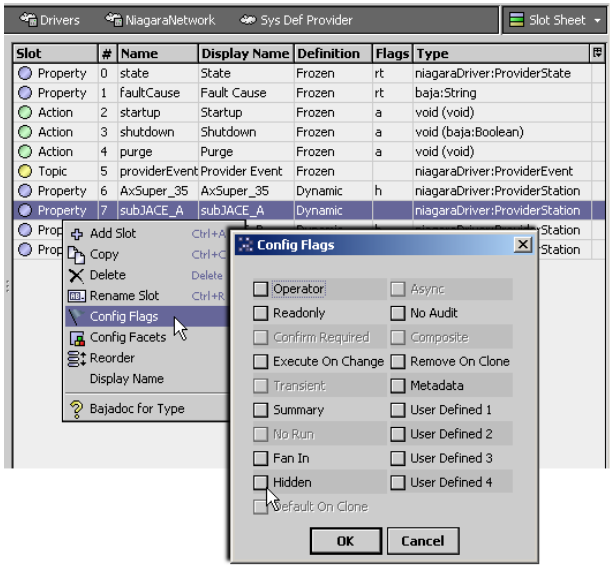


Sys Def Provider (Bog Provider)

State	Running
Fault Cause	

As shown in Figure 2-69, by default the property sheet of a NiagaraNetwork's **`Sys Def Provider`** shows only two status properties: `Status` and `Fault Cause`. However, if you go to its slot sheet you may see additional child "ProviderStation" components.

Figure 2-70 *Sys Def Provider slot sheet, clearing "Hidden" config flag of ProviderStation*



Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	state	State	Frozen	rt	niagaraDriver:ProviderState
Property	1	faultCause	Fault Cause	Frozen	rt	baja:String
Action	2	startup	Startup	Frozen	a	void (void)
Action	3	shutdown	Shutdown	Frozen	a	void (baja:Boolean)
Action	4	purge	Purge	Frozen	a	void (void)
Topic	5	providerEvent	Provider Event	Frozen		niagaraDriver:ProviderEvent
Property	6	AxSuper_35	AxSuper_35	Dynamic	h	niagaraDriver:ProviderStation
Property	7	subJACE_A	subJACE_A	Dynamic		niagaraDriver:ProviderStation
Prop						Station
Prop						Station

☐ Operator	☐ Async
☐ Readonly	☐ No Audit
☐ Confirm Required	☐ Composite
☐ Execute On Change	☐ Remove On Clone
☐ Transient	☐ Metadata
☐ Summary	☐ User Defined 1
☐ No Run	☐ User Defined 2
☐ Fan In	☐ User Defined 3
☐ Hidden	☐ User Defined 4
☐ Default On Clone	

To see child properties of a ProviderStation, you can clear the "Hidden" config flag, as shown being done in Figure 2-70. In turn, the slot sheet of that component also has more hidden properties. Note a child ProviderStation for the local station is also included under any Sys Def Provider (BogProvider).

Note: *An alternative configuration of ProviderStation persistence is possible, using an* **`Orion`** *database instead of station bog storage. That topic is outside the scope of this document.*

Using the hierarchical definitions in the “Role Manager” components of Sys Def device extensions in NiagaraStations (in the NiagaraNetwork of various distributed stations), Sys Def information can be automatically “sync’ed upwards” from the lowest subordinate level stations to “intermediate level” stations. Such stations would be defined as the Supervisor to some stations, but as a subordinate to another station. In this way, “master” Supervisors could have the latest Sys Def information about many more stations than the ones represented locally. This information could be useful in system queries.

Sys Def information includes various connection-type property values, what different services the station is running, and a number of other items,

About the Files extension

Every NiagaraStation has a “**`Files`**” device extension (`NiagaraFileDeviceExt`). It can be used in jobs where files need to be transferred automatically from one station to another, for example, reports from a JACE station to a Supervisor station. Or, a JACE station may import file(s) periodically from a Supervisor station, for some specific reason.

This file import feature is also used in some of the “export tags” functions (for Supervisor auto-configuration) first introduced in AX-3.5. Niagara file import descriptors are automatically created on the Supervisor, for example, when using export tags of type “PxViewTag” in subordinate JACE stations. A separate export tag “FileImportTag” is also available. For details, refer to the *NiagaraAX Export Tags* document.

The **`Files`** device extension has no configuration properties apart from a **`Retry Trigger`** container, (see “About the Retry Trigger” on page 1-34), and otherwise contains child “file import descriptors” that you add using the default view, the **`Niagara File Manager`**.

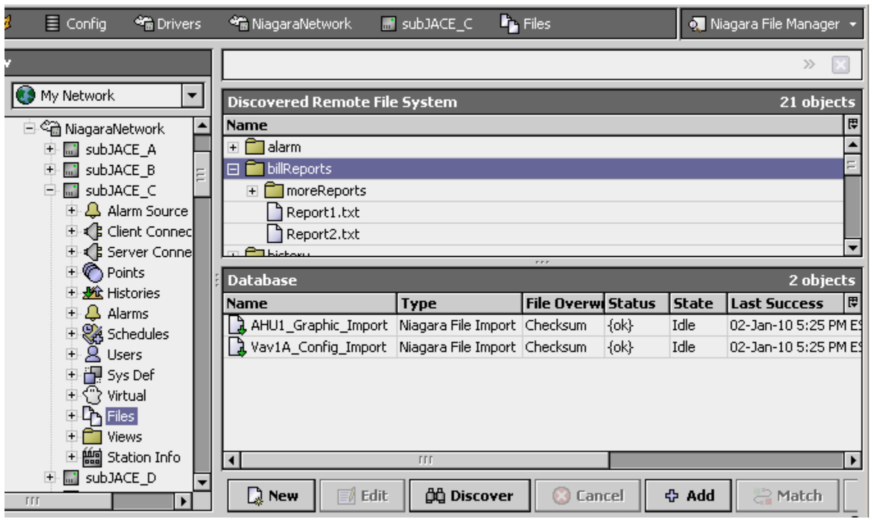
For more details, see the following subsections:

- About the Niagara File Manager
- Add or Edit dialog for NiagaraFileImports
- About Niagara FileImport properties

About the Niagara File Manager

The **`Niagara File Manager`** is the default view of a NiagaraStation’s Files device extension.

Figure 2-71 *Niagara File Manager view of a NiagaraStation’s Files device extension*



Discovered Remote File System — 21 objects

Name
alarm
billReports
moreReports
Report1.txt
Report2.txt

Database — 2 objects

Name	Type	File Overwi	Status	State	Last Success
AHU1_Graphic_Import	Niagara File Import	Checksum	{ok}	Idle	02-Jan-10 5:25 PM E
Vav1A_Config_Import	Niagara File Import	Checksum	{ok}	Idle	02-Jan-10 5:25 PM E

Figure 2-71 shows the view in “learn mode” after a Discover was issued, with files found on the remote station in the top pane, and existing file import descriptors in the lower pane.

Click to select either an entire directory of files to import (as shown selected/highlighted in Figure 2-71), or select a single file to import. If a selected directory, its *entire contents* are imported upon execution—this means all contained files, plus any subdirectories and their contained files.

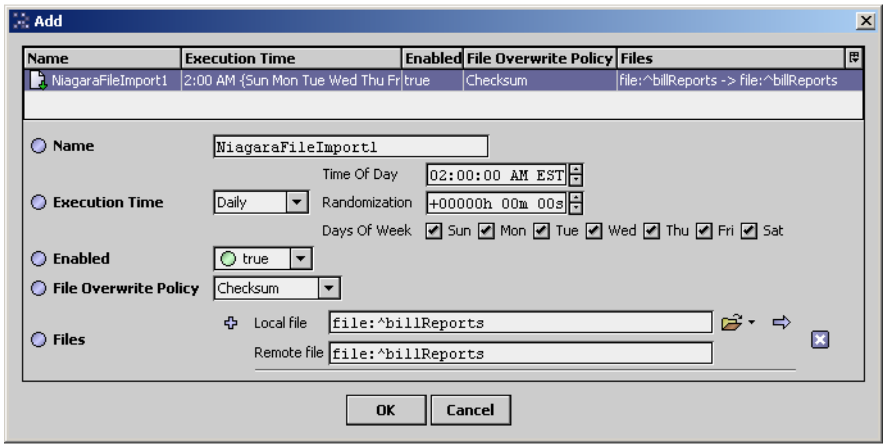
Note: *Hold down the Ctrl key while clicking in the Discovered pane to make multiple selections. Each will have a separate “Files” property to review/edit in the resulting* **`Add`** *dialog for the Niagara file import descriptor.*

With one or more rows in the Discovered pane selected, click the **`Add`** button. The **`Add`** dialog appears. See Add or Edit dialog for NiagaraFileImports.

Add or Edit dialog for NiagaraFileImports

An example **Add** dialog in the Niagara File Manager is shown in Figure 2-72

Figure 2-72 Add dialog for Niagara FileImport in the Niagara File Manager



Name	Execution Time	Enabled	File Overwrite Policy	Files
NiagaraFileImport1	2:00 AM {Sun Mon Tue Wed Thu Fr	true	Checksum	file:^billReports -> file:^billReports

Name	NiagaraFileImport1
Execution Time	Daily; Time Of Day 02:00:00 AM EST; Randomization +00000h 00m 00s; Days Of Week ✓ Sun ✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat
Enabled	true
File Overwrite Policy	Checksum
Files	Local file file:^billReports; Remote file file:^billReports

Properties of file import descriptors available in the **Add** (or **Edit**) dialog are as follows:

- **Name**
  Name for file import descriptor component. Often left at default, i.e. "`NiagaraFileImport`" (appending numerals if needed to keep unique). Editing name does not affect names of imported file(s).
- **Execution Time**
  Either Daily (default), Interval, or Manual. If Manual, properties below are not available:
  - **Time of Day (Daily)**
    Configurable to any daily time. Default is 2:00am.
  - **Randomization (Daily)**
    When the next execution time calculates, a random amount of time between zero milliseconds and this interval is added to the Time of Day. May prevent "server flood" issues if too many file imports are executed at the same time. Default is zero (no randomization).
  - **Days of Week (Daily and Interval)**
    Select (check) days of week for import execution. Default is all days of week.
  - **Interval (Interval)**
    Specifies repeating interval for import execution. Default is every minute.
  - **Time of Day (Interval)**
    Specifies start and end times for interval. Default is 24-hours (start 12:00am, end 11:59pm).
- **Enabled**
  Default is `true`. If set to `false`, file import does not occur.
- **File Overwrite Policy**
  The File Overwrite Policy property determines how existing files are overwritten upon any execution, with two choices available:
  - **Checksum**
    (Default) Checksums for all files are compared for differences. Any files found with differences are re-imported, overwriting the existing files.
  - **Last Modified**
    Date/timestamps for all files are compared. Any remote files found with more recent date/timestamps are re-imported, overwriting the existing files.
- **Files**
  Specifies the selected "Remote file" (or directory) and the "Local file" (or directory), as a pair.
  Note: *Click the ✚. (add) icon, if needed, for additional pairs, or the ☒ (delete) icon to remove a pair.*
  Both fields in any pair should begin with "`file:^`", denoting the root of the station's file space. The two Files fields are:
  - **Local file**
    File path for directory or file to be written in the local station's file space.
    If adding from a Discover, this matches the initially selected "Remote File" value. If local directories used in the file path do not exist, they are created upon import execution.
    To change, you can click one of the right-side controls, either:
    – if a single file, the folder 📂 control for the **File Chooser** dialog.

   - if a directory, the adjacent drop-down control, then choose **Directory Ord Chooser** from the menu for the **Directory Chooser** dialog.

  Or, you may simply edit the file path text, say to add a folder named for the remote station somewhere within the file path.
  For example, from `file:^billReports` to `file:^billReports/subJACE_C`
- **Remote File**
  File path for directory or file to be imported from the remote station's file space. This reflects the selected directory or file for that **Add**. You typically do not edit this value.
  Note: *If a selected directory, the entire contents (all contained files, plus subdirectories and files) are imported upon execution. If already existing, the File Overwrite Policy must be met before each write.*

Additional properties for any file import descriptor are on its property sheet. For details, see "About Niagara FileImport properties".

About Niagara FileImport properties

All properties for any Niagara FileImport descriptor include the following:

- **Status**
  Read-only status of the file import descriptor, typically "ok", unless "disabled" (Enabled=false). A "fault" status may sometimes occur—if so, the reason will be in the **Fault Cause** property.
- **State**
  Current state of the file import, as either `Idle`, `Pending`, or `In Progress`.
- **Enabled**
  Default is `true`. If set to `false`, file import does not occur.
- **Execution Time**
  Either `Daily` (default), `Interval`, or `Manual`.
- **Last Attempt**
  Date/timestamp of the last attempted file import.
- **Last Success**
  Date/timestamp of the last successful file import.
- **Last Failure**
  Date/timestamp of the last failed file import.
- **Fault Cause**
  Typically blank, unless the last import attempt failed. In that case, it provides the reason why the import failed. For example, if the original source file was removed or renamed, it may read similar to:
  `java.lang.Exception: Could not find Remote File: file:^images/tridiumLogo1.gif`
- **Files**
  Specifies a file import target (Local)/source (Remote)"pair" using two fields. Click the ✚. (add) icon, if needed, for additional pairs, or the ☒ (delete) icon to remove a pair.
  For details on this and the next property, see "Add or Edit dialog for NiagaraFileImports".
- **File Overwrite Policy**
  The File Overwrite Policy property determines the criteria used for overwriting existing files upon any execution, with two choices: `Checksum` (default) or `Last Modified`.

CHAPTER 3

Field Bus Integrations

For purposes here, a field bus integration is any NiagaraAX driver *besides* the niagaraDriver (Niagara Network). All Niagara AX drivers resemble each other in basic architecture, including the Niagara Network. For more details, see "About Network architecture" on page 1-2, and "About the Niagara Network" on page 2-1.

Field bus integrations such as BACnet, LON, Modbus, as well as various "legacy" drivers (typically serial-connected) each have unique characteristics and features. This section provides a collection of topics that apply to some of these drivers.

The following main sections are included:

- Port and protocol variations
- Learn versus New devices and points
- Serial tunneling

Port and protocol variations

With one exception, each field bus driver (network) associates with only *one* physical communications port on the host platform, and uses a specific communications protocol. The exception is the BACnet driver (BacnetNetwork), where multiple ports (Ethernet, RS-485 for MS/TP) and protocol variants (BACnet-Ethernet, BACnet-IP, BACnet-MS/TP) may be used. See the *Bacnet Guide* for details.

Generally, field bus drivers can be categorized as one of the following:

- Ethernet-connected driver
- Serial-connected driver
- Special-port driver

Ethernet-connected driver

Many field bus drivers are Ethernet (port) connected, typically using some TCP/IP protocol for transport. For example, the Modbus TCP driver (ModbusTcpNetwork) uses the Modbus TCP protocol—essentially the Modbus protocol "wrapped" in TCP/IP. The SNMP driver (SnmpNetwork) uses SNMP, an application-layer protocol within the TCP/IP protocol suite. These and other Ethernet-connected drivers operate from a single Ethernet port without difficulty, due to the high bandwidth and efficiencies of IEEE 802 network (and TCP/IP) standards.

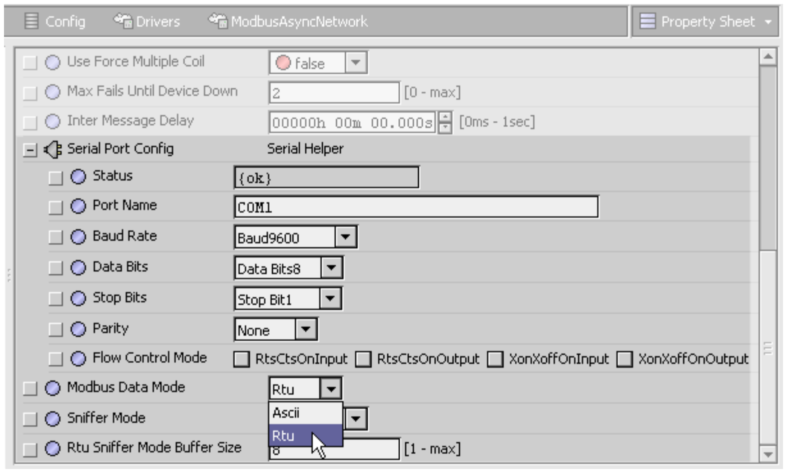
In addition to JACE platform usage, Ethernet-connected drivers are available for the Supervisor (PC) platform as well, for "direct device integrations." These are specially-licensed versions of the Supervisor (by default, a Supervisor is licensed only for JACE device communications, via the NiagaraNetwork).

Serial-connected driver

Serial-connected drivers use a *specific* serial port on the host (JACE) platform. For example, the Modbus serial driver (ModbusAsyncNetwork) requires association with a specific `COMn` port on the JACE, which you do from the property sheet of this network component (Figure 3-1).

Note: *Only one network can be assigned to any one serial port (`COMn`) of the host JACE platform. That driver network essentially "owns" that communications port.*

Figure 3-1 *Serial port configuration for a serial-based driver*



Config | Drivers | ModbusAsyncNetwork | Property Sheet

```
Use Force Multiple Coil        false
Max Fails Until Device Down    2                 [0 - max]
Inter Message Delay            00000h 00m 00.000s  [0ms - 1sec]
Serial Port Config             Serial Helper
  Status                       {ok}
  Port Name                    COM1
  Baud Rate                    Baud9600
  Data Bits                    Data Bits8
  Stop Bits                    Stop Bit1
  Parity                       None
  Flow Control Mode            RtsCtsOnInput  RtsCtsOnOutput  XonXoffOnInput  XonXoffOnOutput
Modbus Data Mode               Rtu
Sniffer Mode                   Ascii
                               Rtu
Rtu Sniffer Mode Buffer Size   8                 [1 - max]
```

Slots under the “Serial Port Config” (SerialHelper) must be set to match the communications parameters of other devices on the attached network. Note that in this ModbusAsync example, you also select either the Modbus ASCII or Modbus RTU protocol (the driver supports either one, which you set according to the type of networked Modbus devices).

Often, serial-connected drivers support “legacy type” device networks. In this case, the “serial tunneling” feature may be useful to run vendor-specific legacy Windows applications to do device configuration and maintenance (all from an IP station connection). See “Serial tunneling” on page 3-2.

Special-port driver

JACE controllers may include one or more special-use ports, for example one or more Echelon (LON) FTT-10 ports. Usage of such a port requires a specific driver. In this example, the Lonworks driver (each LonNetwork) associates with a specific `LONn` port, which is configured under that network component. For details, see the *Lonworks Guide*.

Other special-use ports may appear as the evolution of JACE products continue.

Learn versus New devices and points

Many, if not most, field bus drivers provide the ability to “learn” devices and data points while connected to that particular field bus. Exceptions include drivers where the field bus protocol does not provide the ability for this, for example, any of the Modbus drivers.

For specific learn procedures, see the “Quick Start” section in the various driver documents.

Serial tunneling

A NiagaraAX station running one or more serial-based drivers can provide “tunneling” access to its connected devices. This allows you to use a vendor’s Windows serial-based application (via the serial tunnel client) to perform device-specific operations. Examples include application downloads or other device configuration.

The tunneling client is separate from NiagaraAX Workbench—meaning that you can install it on various PCs, as needed. The key advantage is that serial tunneling requires only a standard IP connection (to the station), yet provides access as if the client PC was attached to the target serial network via a physical COM port, for example RS-232.

Note: *No special licensing is required to use tunneling features in NiagaraAX.*

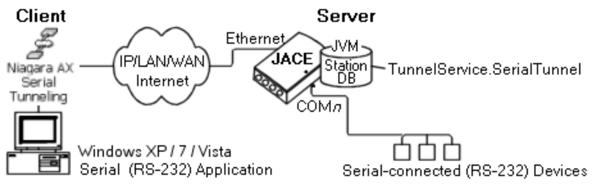
The following sections provide more details:

- Serial tunnel overview
- Client side (PC application)
- Station side (TunnelService)
- Serial tunneling usage notes

Serial tunnel overview

As shown in Figure 3-2, tunneling in NiagaraAX uses a client-server architecture.

Figure 3-2 *Serial tunnel architecture*


Client
Server
Niagara AX Serial Tunneling
IP/LAN/WAN Internet
Ethernet
JACE
JVM
Station DB
TunnelService.SerialTunnel
COMn
Windows XP / 7 / Vista Serial (RS-232) Application
Serial-connected (RS-232) Devices

Serial tunneling uses the following components:

- **Client (PC application) side**
  The **`NiagaraAX Serial Tunnel`** client installs on most Windows-based PCs (independent of Niagara Workbench). It provides a "Niagara AX Serial Tunneling" client via a "virtual" `COMn` port. For details, see "Client side (PC application)" on page 3-3.
- **Server (station) side**
  The host JACE must have the "`tunnel`" module installed to support serial tunneling. In addition, its station must be configured with a **`TunnelService`** and a child **`SerialTunnel`** component. For details, see "Station side (TunnelService)" on page 3-7.

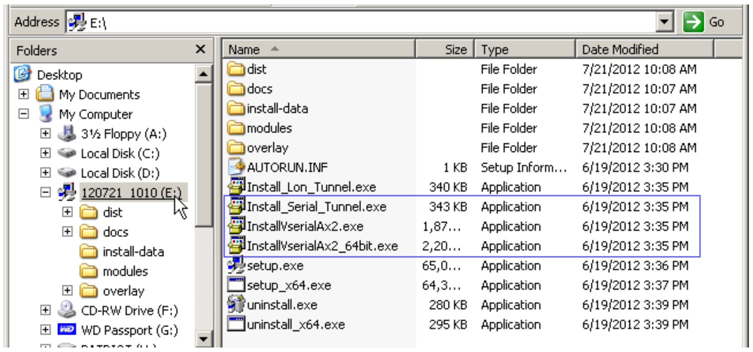
Note: *A LonTunnel ("`lontunnel`" module) is also available, and uses the same basic architecture—as a child under a station's TunnelService. The LonTunnel allows a Lonworks application tunneling to connected LON devices on a JACE's FTT-10 network. For details, see "Lon tunneling" in the* Lonworks Guide.

Note client tunnel connections (serial or Lon) to a station's TunnelService use basic authentication *(creden-tials* not *encrypted), so security considerations apply. See "Best security practices for tunneling" on page 3-8.*

Client side (PC application)

The serial tunnel client is a self-installing executable found in the root of the NiagaraAX distribution CD. There are three different versions of the executable, as shown in Figure 3-3 below.

Figure 3-3 *Serial Tunnel Client installation file is in NiagaraAX CD root*


Address E:\ Go

Folders
- Desktop
  - My Documents
  - My Computer
    - 3½ Floppy (A:)
    - Local Disk (C:)
    - Local Disk (D:)
    - 120721_1010 (E:)
      - dist
      - docs
      - install-data
      - modules
      - overlay
    - CD-RW Drive (F:)
    - WD Passport (G:)

Name	Size	Type	Date Modified
dist		File Folder	7/21/2012 10:08 AM
docs		File Folder	7/21/2012 10:07 AM
install-data		File Folder	7/21/2012 10:07 AM
modules		File Folder	7/21/2012 10:08 AM
overlay		File Folder	7/21/2012 10:08 AM
AUTORUN.INF	1 KB	Setup Inform...	6/19/2012 3:30 PM
Install_Lon_Tunnel.exe	340 KB	Application	6/19/2012 3:35 PM
Install_Serial_Tunnel.exe	343 KB	Application	6/19/2012 3:35 PM
InstallVserialAx2.exe	1,87...	Application	6/19/2012 3:35 PM
InstallVserialAx2_64bit.exe	2,20...	Application	6/19/2012 3:35 PM
setup.exe	65,0...	Application	6/19/2012 3:36 PM
setup_x64.exe	64,3...	Application	6/19/2012 3:37 PM
uninstall.exe	280 KB	Application	6/19/2012 3:39 PM
uninstall_x64.exe	295 KB	Application	6/19/2012 3:39 PM

Depending on your client PC's Windows operating system, select to install from *one* of the following:

- Windows XP: `Install_Serial_Tunnel.exe`
- Windows 7 or Windows Vista (32-bit): `InstallVserialAx2.exe`
- Windows 7 or Windows Vista (64-bit): `InstallVserialAx2_64bit.exe`

See the following additional sections for more details:

- Installing the serial tunnel client
- Serial tunnel client configuration
- Serial tunnel client installation details

Installing the serial tunnel client

To install the serial tunnel client on a Windows machine

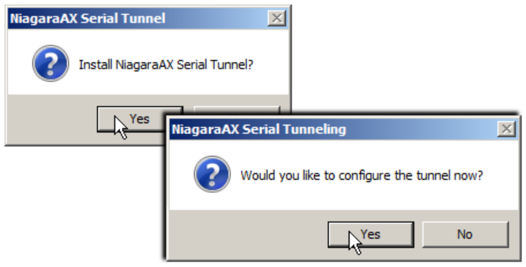
To install the serial tunnel client, at the Windows PC do the following:

Step 1 Access the appropriate installation executable, as found in the root of the NiagaraAX CD (Figure 3-3).

- Windows XP: `Install_Serial_Tunnel.exe`
- Windows 7 or Windows Vista (32-bit): `InstallVserialAx2.exe`

- Windows 7 or Windows Vista (64-bit): `InstallVserialAx2_64bit.exe`

Step 2 Double-click this file to launch the installation.


NiagaraAX Serial Tunnel
Install NiagaraAX Serial Tunnel?
NiagaraAX Serial Tunneling
Would you like to configure the tunnel now?

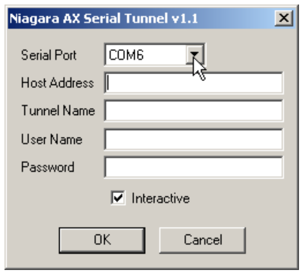
Click **Yes** to install, where other popup appears, as shown above.

Click **Yes** again to configure.

Step 3 In the **Niagara AX Serial Tunnel** dialog, enter any known data, or accept defaults.

- In the Windows XP serial tunnel client dialog, you select a serial port (COM*n*), as shown in Figure 3-4.

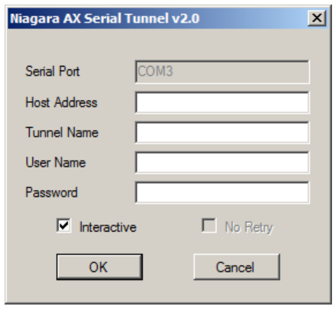
Figure 3-4 *Example Windows XP serial tunnel client configuration defaults*


Niagara AX Serial Tunnel v1.1
Serial Port: COM6
Host Address
Tunnel Name
User Name
Password
[✓] Interactive

Do not duplicate any existing serial (COM*n*) port already used by Windows. You can always reconfigure again, by returning via the Windows XP **Control Panel**. If left "interactive" all remaining fields in this dialog are editable each time you specify this COM port from the serial application that you are tunneling (this popup dialog reappears each time).

- In the Windows 7 / Vista serial tunnel client dialog, an unused port (COM*n*) is *already selected*, as shown in Figure 3-4

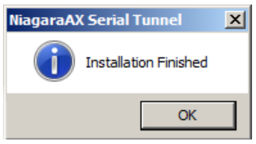
Figure 3-5 *Example Windows7/Vista serial tunnel client configuration defaults*


Niagara AX Serial Tunnel v2.0
Serial Port: COM3
Host Address
Tunnel Name
User Name
Password
[✓] Interactive
[ ] No Retry

If left 'interactive", you can configure the remaining fields in this dialog each time you specify this COM port from your serial application you are tunneling (this popup dialog reappears each time).

See "Serial tunnel client configuration" on page 3-5 for details on all fields in this dialog. Parameters work essentially the same whether you have the Windows XP driver or (either) Windows 7/Vista driver.

Step 4 Click **OK** to finish the install.


NiagaraAX Serial Tunnel
Installation Finished

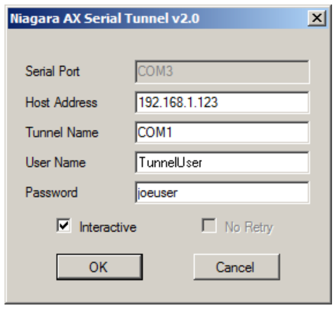
The "Installation Finished" dialog appears—click **OK** again. See "Serial tunnel client installation details" on page 3-5 for a listing of installed components.

Serial tunnel client configuration

By default after installation you see the serial tunnel configuration dialog each time you specify its named Serial Port (COM*n*) from your serial application. In the case of the Windows XP driver, you can also access this dialog by selecting "NiagaraAX Serial Tunneling Client" from the Windows XP **Control Panel**.

As shown for an example session in Figure 3-6, all tunnel client fields require a valid entry.

Figure 3-6 *Serial tunnel client session example*


Niagara AX Serial Tunnel v2.0

Serial Port: COM3
Host Address: 192.168.1.123
Tunnel Name: COM1
User Name: TunnelUser
Password: joeuser
Interactive: ✓
No Retry: ☐

Fields in this dialog are described as follows:

- **Serial Port**
  The "virtual" COM port provided by this tunnel client. This should not conflict with any existing COM port assignment, as known to Windows, for a physical serial port (e.g. COM1).
  When you tunnel from a serial-based Windows application, you specify this "virtual" COM port.
- **Host Address**
  The IP address (or hostname) of the tunnel server, meaning the target JACE running a station with a serial-based network, TunnelService, and SerialTunnel.
- **Tunnel Name**
  The COMn device name (identifier) of the JACE's driver network to access. This will vary depending on the configuration of the network and its corresponding SerialTunnel.
- **User Name**
  User in the target JACE station, where this station user must have *admin write* permissions for the station's TunnelService and child SerialTunnel(s).

Caution *The TunnelService uses "basic authentication" for login on any client connection (serial tunnel or Lon tunnel), so we recommend you create a special user in the station to use (only) for all serial or Lon tunnel access. For configuration details, see "Best security practices for tunneling" on page 3-8.*

- **Password**
  Password for this station user.
  Note: *The password is* not encrypted *when passed to the station (see Caution above).*
- **Interactive (checkbox)**
  If checked, this dialog reappears each time a serial-based application first opens this "virtual" COM port. If cleared, this dialog displays only if an open fails to establish a connection to the tunnel server (as stored from last entry). Typically, you leave Interactive checked (regardless of driver version).
  - In the case of the Windows XP driver, when this dialog appears interactively, the **Serial Port** setting is *read-only*. To change it, you must access the Serial Tunneling applet from the Windows XP **Control Panel**.
  - In the case of either Windows 7/Vista driver, the Serial Port setting is always read only. However, a second "No Retry" checkbox becomes available if you clear "Interactive".

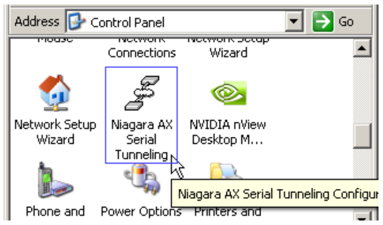
Serial tunnel client installation details

Depending which type of serial tunnel client you installed, the driver's Windows interface, files installed, and method of uninstalling vary.

- Windows XP serial tunnel client details
- Windows 7 / Vista serial tunnel client details

Windows XP serial tunnel client details The Windows XP serial tunnel client installs as a Windows service (**NiagaraAX Serial Tunnel**), and has a Control Panel applet available (Figure 3-7).

Figure 3-7 *Windows XP control panel applet for NiagaraAX serial tunnel client*



The following files are installed, with services referenced in the Windows registry:

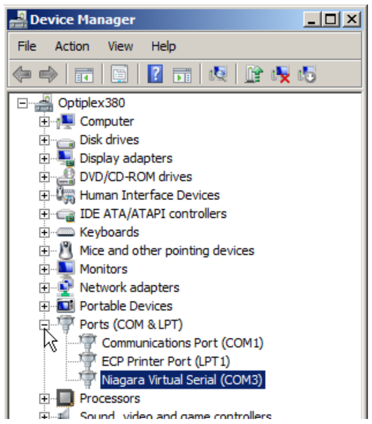
- Control Panel
  <WINDOWS_SYSTEM_DIR>`\vserax.cpl`
- Network Tunnel Service
  <WINDOWS_SYSTEM_DIR>`\vserax.exe`
  Service name: `vseraxSvc` (`vserax` dependency)
- Serial Driver Service
  <WINDOWS_SYSTEM_DIR>`\drivers\vseraxx.sys`
  Service name: `vserax`
- Uninstaller
  <WINDOWS_SYSTEM_DIR>`\vseraxun.exe`

If necessary, uninstall the serial tunnel client driver using the "Add or Remove Programs" entry from the Windows XP **`Control Panel`**.

Note: *If uninstalling, and the uninstall appears to fail, try reinstalling the tunnel client, and then uninstall again.*

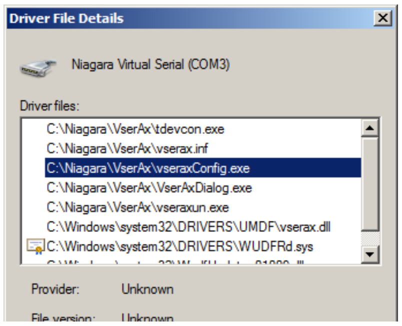
Windows 7 / Vista serial tunnel client details The serial tunnel client for Windows 7/Vista installs as a "virtual" serial COM port (no separate Windows service or Control Panel applet). You can this port listed in the Windows **`Device Manager`** under **`Ports`** (Figure 3-8).

Figure 3-8 *Windows 7 / Vista "virtual" COM port for the NiagaraAX serial tunnel client*



From the Windows **`Device Manager`**, you can right-click this virtual COM port to see its **`Properties`**, or if necessary, to *Uninstall* the driver. From its **`Properties`** dialog, the "Driver" tab and "Driver Details" button provides a popup dialog that lists the various files installed (Figure 3-9).

Figure 3-9 *Driver file details for NiagaraAX serial tunnel client (Windows 7 / Vista)*



C:\Niagara\VserAx\tdevcon.exe
C:\Niagara\VserAx\vserax.inf
C:\Niagara\VserAx\vseraxConfig.exe
C:\Niagara\VserAx\VserAxDialog.exe
C:\Niagara\VserAx\vseraxun.exe
C:\Windows\system32\DRIVERS\UMDF\vserax.dll
C:\Windows\system32\DRIVERS\WUDFRd.sys

Provider: Unknown
File version: Unknown

The file shown highlighted (`vseraxConfig.exe`) produces the tunnel client's configuration dialog.

Station side (TunnelService)

To be a serial "tunnel server," a JACE must have the `tunnel` module installed, and its station must have a TunnelService (under its **`Services`** folder), as well as a child SerialTunnel component.

The following sections provide more details on the "station side" of serial tunneling:

- Configuring the serial tunnel server
- Best security practices for tunneling
- About serial tunnel connections

Configuring the serial tunnel server

To configure the station for serial tunneling

To configure the station for serial tunneling, do the following:

Step 1 In the palette side bar, open the **`tunnel`** palette.

Step 2 Open the JACE station and expand its **`Services`** folder.

- If no **`TunnelService`** exists, paste one from the palette into the **`Services`** folder.
- If a TunnelService does exist, go to next step.

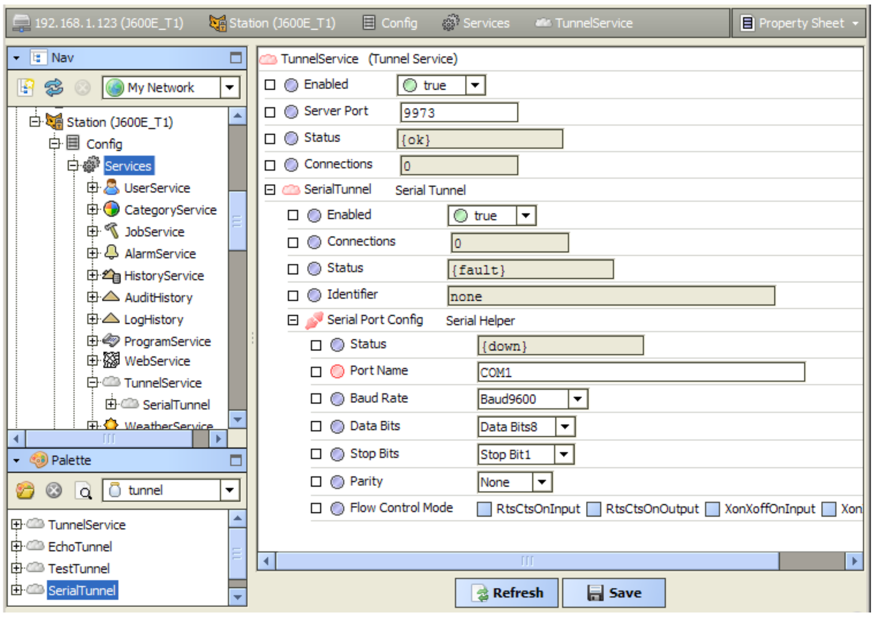
Note: *Only one TunnelService is needed in a station's services, and it can hold multiple tunnel components (SerialTunnel and LonTunnel). The TunnelService in the (serial)* **`tunnel`** *module is identical to the TunnelService in the* **`lontunnel`** *module.*

Step 3 From the palette, paste a **`SerialTunnel`** under the station's **`TunnelService`**.

The station should now have a TunnelService and a child SerialTunnel component.

Step 4 In the SerialTunnel's property sheet, expand the "Serial Port Config container" (Figure 3-10).

Figure 3-10 *TunnelService with child SerialTunnel (copied from tunnel palette)*



```
TunnelService  (Tunnel Service)
  Enabled            true
  Server Port        9973
  Status             {ok}
  Connections        0
  SerialTunnel       Serial Tunnel
    Enabled          true
    Connections      0
    Status           {fault}
    Identifier       none
    Serial Port Config   Serial Helper
      Status         {down}
      Port Name      COM1
      Baud Rate      Baud9600
      Data Bits      Data Bits8
      Stop Bits      Stop Bit1
      Parity         None
      Flow Control Mode   RtsCtsOnInput  RtsCtsOnOutput  XonXoffOnInput  Xon
```

Here, type in the `COMn` "Port Name" used by the target driver network, and specify other parameters as defined in the network's configuration. Port Name should be similar to `COM1` or `COM2`, and so on.

See SerialHelper on page 11 for details on various properties.

Note: *If the JACE has multiple serial-type driver networks (and corresponding COM ports), you can copy the same number of SerialTunnel components under the TunnelService. You can then associate each Serial-Tunnel with a particular network (by its COMn port name), and set the other parameters accordingly.*

Step 5 **`Save`** the **`TunnelService`** configuration when done.

A station user requires *admin write* permissions for the SerialTunnel(s) to allow tunnel client access.

Note: *Clients that access the TunnelService (both SerialTunnel and LonTunnel) use "basic authentication" for login access. So in either of these client connections, the user credentials passed to the station are not encrypted—a potential security issue! See "Best security practices for tunneling" on page 3-8.*

Also, consider disabling *the TunnelService (set its* Enabled *property to* `false`*) whenever it is not needed.*

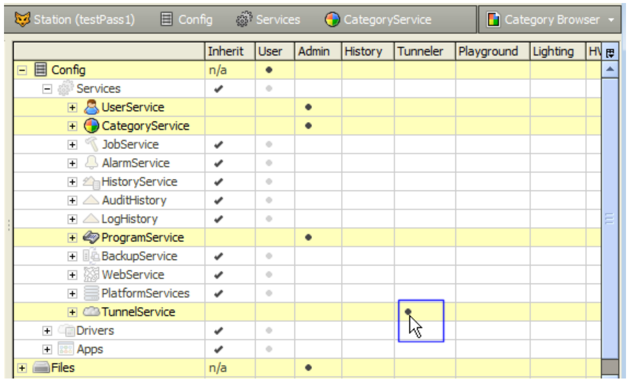
Best security practices for tunneling

Although convenient, serial or Lon tunneling access to a station presents a potential security issue, as a station's **`TunnelService`** uses "basic authentication" for client access to the station. This differs from normal user access (via FoxService and/or WebService), typically using much stronger authentication.

As a workaround, we strongly recommend that you assign the station's TunnelService to a special category *not assigned to any other component* in the station, and create a *special user* that has admin write permissions on *only that single category* (unlike any other user). That should be the *only user* used to make tunnel client connections. See "To configure for safer tunneling access".

To configure for safer tunneling access

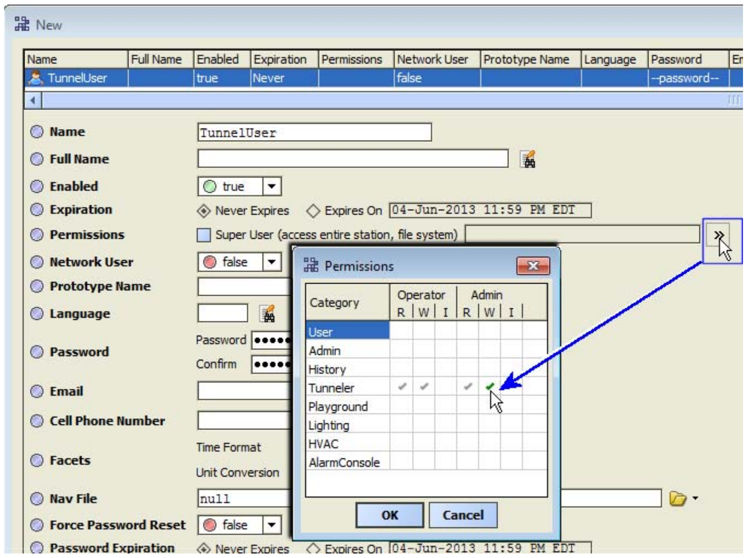
Step 1 In the station's **`CategoryService`**, set up a Category *unassigned to any other component.*



	Inherit	User	Admin	History	Tunneler	Playground	Lighting	HV
Config	n/a	●						
Services	✓	●						
UserService			●					
CategoryService			●					
JobService	✓	●						
AlarmService	✓	●						
HistoryService	✓	●						
AuditHistory	✓	●						
LogHistory	✓	●						
ProgramService			●					
BackupService	✓	●						
WebService	✓	●						
PlatformServices	✓	●						
TunnelService					●			
Drivers	✓	●						
Apps	✓	●						
Files	n/a		●					

Assign the station's **`TunnelService`** to that category, as shown above.

Step 2 In the station's **`UserService`**, create a new user that has permissions *only* on that one category.



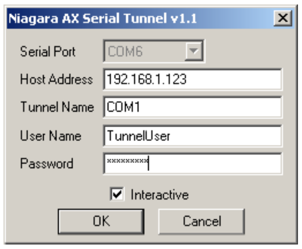
Name	Full Name	Enabled	Expiration	Permissions	Network User	Prototype Name	Language	Password	En
TunnelUser		true	Never		false			--password--	

Name	TunnelUser
Full Name	
Enabled	true
Expiration	[✓ Never Expires] Expires On 04-Jun-2013 11:59 PM EDT
Permissions	Super User (access entire station, file system)
Network User	false
Prototype Name	
Language	
Password	Password ●●●●● Confirm ●●●●●
Email	
Cell Phone Number	
Facets	Time Format Unit Conversion
Nav File	null
Force Password Reset	false
Password Expiration	[✓ Never Expires] Expires On 04-Jun-2013 11:59 PM EDT

Category	Operator R	Operator W	Operator I	Admin R	Admin W	Admin I
User						
Admin						
History						
Tunneler	✓	✓		✓	✓	
Playground						
Lighting						
HVAC						
AlarmConsole						

Assign this new user *admin write* permissions to that *one* category, and **`Save`** that user.

Step 3 From any client to the TunnelService (serial tunnel *or* Lon tunnel), only use this special user account.



Serial Port	COM6
Host Address	192.168.1.123
Tunnel Name	COM1
User Name	TunnelUser
Password	*********
☑ Interactive	

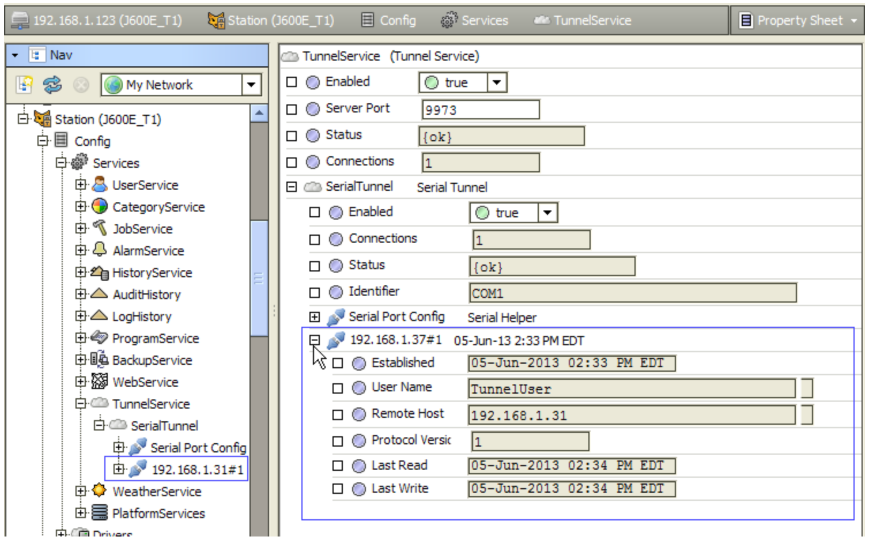
This workaround provides full tunneling capability, but minimizes the security risk in case the credentials for this special user become compromised.

About serial tunnel connections

Under any SerialTunnel, only one tunnel connection is supported at any one time—if a tunnel connection is active and another tunnel client (PC) attempts to connect, that remote user sees a popup dialog saying that the "Desired tunnel is busy."

In the station (tunnel server), any active tunnel connection results in a TunnelConnection child component, named as the remote (client's) IP address or hostname, with a "`#1`" suffix (Figure 3-11).

Figure 3-11 *TunnelConnection is dynamically created/removed*



TunnelService (Tunnel Service)
Enabled true
Server Port 9973
Status {ok}
Connections 1
SerialTunnel Serial Tunnel
Enabled true
Connections 1
Status {ok}
Identifier COM1
Serial Port Config Serial Helper
192.168.1.37#1 05-Jun-13 2:33 PM EDT
Established 05-Jun-2013 02:33 PM EDT
User Name TunnelUser
Remote Host 192.168.1.31
Protocol Versic 1
Last Read 05-Jun-2013 02:34 PM EDT
Last Write 05-Jun-2013 02:34 PM EDT

In the Figure 3-11 example, the remote host that is currently serial tunneling is "`192.168.1.31`." When a tunnel connection is terminated, this Tunnel Connection component is removed.

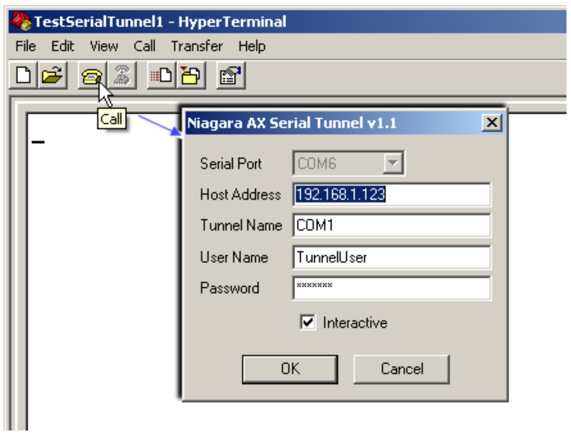
In addition to its statistical properties, a TunnelConnection has an available Disconnect action. This disconnects the active tunnel connection, removing the parent TunnelConnection component. A popup dialog "Connection closed by remote host" is seen on the client tunnel side.

Serial tunneling usage notes

Serial tunneling may not work with all vendor's serial-connected Windows applications. Also, serial tunneling is not supported for BACnet MS/TP usage—however, BACnet *router* functionality provided by a station running a BacnetNetwork with multiple network ports (e.g. IpPort, MstpPort) provides IP access to MS/TP connected devices.

When initiating a connection through the serial tunnel, client-side usage is transparent except for the "virtual `COMn`" port used, and if "Interactive" is left enabled (typical) the resulting serial tunnel configuration dialog right before the connection is attempted. Figure 3-12 shows an example serial connection being initiated from the Windows Hyperterminal application. Tunneling client messages may appear if the connection fails.

Figure 3-12 *Example Windows XP serial application (Hyperterminal) initiating tunnel connection*


TestSerialTunnel1 - HyperTerminal
File Edit View Call Transfer Help
Call
Niagara AX Serial Tunnel v1.1
OK
Cancel

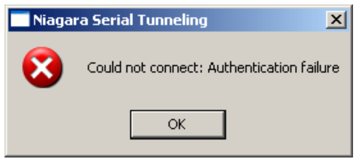
Serial Port COM6
Host Address 192.168.1.123
Tunnel Name COM1
User Name TunnelUser
Password *******
☑ Interactive

Speed of the tunnel connection may be slower that a direct serial connection, due to the overhead of wrapping and unwrapping messages in Niagara Fox and TCP/IP protocols.

Tunneling client messages

When using a tunnel client, the specified user must be "authenticated" by the station before a connection is granted (established). If the user is not found, or if the entered password is incorrect, a popup message may appear on the client PC (Figure 3-13). Note the User Name and Password are both case-sensitive.

Figure 3-13 *Authentication issue*

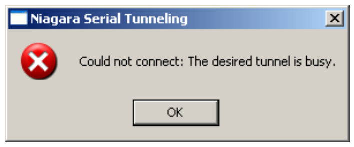


Could not connect: Authentication failure

Caution *As previously cautioned, note that "basic authentication" is used in any client connection to the station's TunnelService, with possible security consequences. See "Best security practices for tunneling" on page 3-8.*

Currently, only one tunnel connection is allowed per SerialTunnel. If another client application attempts a connection to that tunnel, a popup message may appear on that PC (Figure 3-14).

Figure 3-14 *Tunnel busy*



Could not connect: The desired tunnel is busy.

CHAPTER 4

Plugin Guides

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.

Types of modules with plugins

Following, is a list of modules with driver-related plugins (views):

- Plugins in driver module
- Plugins in niagaraDriver module
- Plugins in niagaraVirtual module

Plugins in driver module

- driver-DelimitedFileImportManager
- driver-DeviceManager
- driver-DriverManager
- driver-FileDeviceManager
- driver-HistoryExportManager
- driver-HistoryImportManager
- driver-PointManager

driver-DelimitedFileImportManager

The **`Delimited File Import Manager`** view is the default view on the **`Histories`** extension (FileHistoryDeviceExt) of a FileDevice (in a FileNetwork). See Figure 5-1 on page 4.

This manager view is similar to other driver's history import managers—see "History Import Manager" on page 1-46. However, there is no "Discover" feature to add history file import descriptors (types ExcelCsvFileImport and DelimitedFileImport). Instead, you use the History Import New function to add history file import descriptors, as each is simply to reference a local delimited-type text file (such as a CSV type).

Like other manager views, the Delimited File Import Manager view is a table-based view, where you create (**`New`**) history file import descriptors, or double-click existing import descriptors to make History Import Edits. For details, see the next section "Delimited File Import Manager usage notes".

Delimited File Import Manager usage notes The following notes apply specifically to using the DelimitedFileImportManager view, the view most used within a FileNetwork (outside of property sheets for created history file import descriptors).

- When adding a new descriptor, 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.
- Add a *single* descriptor referencing a File that is patterned like other delimited files you wish to import, then 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. Then, *duplicate* that working import descriptor, (and before executing) change its File reference, History Id, and whatever other properties you wish to keep unique. This saves engineering time. For related details, see "Properties of history file import descriptors" on page 5-2.

driver-DeviceManager

The Device Manager plugin allows you to create and manage devices. The DeviceManager is a view on a network component in a station. For details, see “About the Device Manager” on page 1-14.

driver-DriverManager

The Driver Manager plugin allows you to add and view drivers. It is available on the DriverContainer. For more details, see “About the Driver Manager” on page 1-4.

driver-FileDeviceManager

The File Device Manager is the default view on a FileNetwork. For general information, see “About the Device Manager” on page 1-14. See the section “driver-FileNetwork” on page 5-4 for more details about a File Network. Also, note that the view used most often under a File Network is the DelimitedFileImportManager view of the Histories extension of a FileDevice.

driver-HistoryExportManager

History Export Manager provides a view of NiagaraHistoryDeviceExt. To see the view, right-click on a NiagaraHistoryDeviceExt in a Station and select **`Views > History Export Manager`**. For details, see “Niagara History Export Manager” on page 2-33.

driver-HistoryImportManager

History Import Manager provides a view of NiagaraHistoryDeviceExt. To see the view, right-click on a NiagaraHistoryDeviceExt in a Station and select **`Views > HistoryImportManager`**. For details, see “History Import Manager” on page 1-46.

driver-PointManager

Point Manager provides an overview of the proxy points mapped into the PointDeviceExt. In a NiagaraNetwork, PointManager is a view on the NiagaraPointDeviceExt of a NiagaraStation. To see the PointManager, right-click on a Points device extension and select **`Views > Point Manager`**. For details, see “About the Point Manager” on page 1-37.

Plugins in niagaraDriver module

- NiagaraFileManager
- NiagaraHistoryExportManager
- NiagaraHistoryImportManager
- NiagaraPointManager
- NiagaraScheduleExportManager
- NiagaraScheduleImportManager
- ServerConnectionsSummary
- StationManager
- UserSyncManager

niagaraDriver-NiagaraFileManager

`Niagara File Manager` is the default view of a NiagaraStation’s “**`Files`**” device extension (NiagaraFileDeviceExt). To see the view, double-click on this extension, or right-click and select **`Views > Niagara File Manager`**. For details, “About the Niagara File Manager” on page 2-54.

niagaraDriver-NiagaraHistoryExportManager

Niagara History Export Manager provides a view of NiagaraHistoryDeviceExt. To see the view, right-click on a NiagaraHistoryDeviceExt in a Station and select **`Views > Niagara History Export Manager`**. For more details, see “Niagara History Export Manager” on page 2-33.

niagaraDriver-NiagaraHistoryImportManager

Niagara History Import Manager provides a view of NiagaraHistoryDeviceExt. To see the view, Right-click on a NiagaraHistoryDeviceExt in a Station and select **`Views > Niagara History Import Manager`**. For more details, see “History Import Manager” on page 1-46.

niagaraDriver-Niagara PointManager

Niagara Point Manager provides access to the proxy points mapped into the PointDeviceExt. The Niagara PointManager is a view on the NiagaraPointDeviceExt in a Station in a NiagaraNetwork. For general information, see “About the Point Manager” on page 1-37. For specific details, see “Niagara Point Manager notes” on page 2-20.

niagaraDriver-NiagaraScheduleExportManager

The **Niagara Schedule Export Manager** is used to manage schedules exported from a NiagaraStation. For more details, see "Schedule Export Manager" on page 1-53, and also "Station Schedules import/export notes" on page 2-28.

niagaraDriver-NiagaraScheduleImportManager

The **Niagara Schedule Import Manager** is used to manage schedules imported from a NiagaraStation. For more details, see "Schedule Import Manager" on page 1-49, and also "Station Schedules import/export notes" on page 2-28.

niagaraDriver-ServerConnectionsSummary

The **Server Connections Summary** is the default view of the ServerConnections slot in the Niagara Fox Service, under the NiagaraNetwork. It provides a table listing *current* client connections to the station's Fox server (station-to-station connections are not included).

Note: *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.*

Included in connections summary table are the following columns:

- **Address**
  IP address of the Fox client connected to the station, along with its remote TCP port.
- **User**
  Station user account used for authentication.
- **Connection Time**
  Timestamp of when the Fox connection occurred.
- **Application**
  Client software used to access the station (for example, `Niagara Workbench 3.0.76`).

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 **SessionN**, with status slots of historical information, including connect and disconnect entries.

Access the **Server Connections Summary** from the *property sheet* of the NiagaraNetwork. Expand the **Fox Service**, then click the **Server Connections** slot.

niagaraDriver-StationManager

The **Station Manager** view on the NiagaraNetwork allows you to manage and access NiagaraStations. For more details, see "Niagara Station Manager notes" on page 2-9.

niagaraDriver-UserSyncManager

The **User Sync Manager** view on the NiagaraNetwork allows you to manage the Users device extension properties for all NiagaraStations in the network. For more details, see "About the User Sync Manager" on page 2-18.

Plugins in niagaraVirtual module

- Niagara Virtual Cache View
- Nva File View

niagaraVirtual-NiagaraVirtualCacheView

Niagara Virtual Cache View is the default view on the "Cache" component child of the "Virtual Policies" container of a NiagaraNetwork. For more details, see "Niagara virtuals cache (Virtual Policies)" on page 2-42.

niagaraVirtual-Nva File View

Nva File View is a view of a station's "Niagara virtual archive" cache file (e.g. `cache1.nva`) when accessed from Workbench. For related details, see "Virtual Policies (Cache) properties" on page 2-44.

Component Guides

These Component Guides provides summary information on components commonly used in drivers.

Component Reference Summary

Summary information is provided on components in the following modules:

- driver
- fox
- niagaraDriver
- serial
- tunnel

Components in driver module

- ArchiveFolder
- ConfigRule
- DelimitedFileImport
- DriverContainer
- ExcelCsvFileImport
- FileDevice
- FileHistoryDeviceExt
- FileHistoryWorker
- FileNetwork
- HistoryNetworkExt
- PingMonitor
- SendTimer
- SendTimes
- TuningPolicy
- TuningPolicyMap

driver-ArchiveFolder

ArchiveFolder is a folder available to hold and organize history "import descriptors" and/or "export descriptors". You can add such folders using the **New Folder** button in the **History Import Manager** view or **History Export Manager** view of the "Histories" extension of a device component (NiagaraStation, FileDevice, various "rdbmsDatabase" components). Each ArchiveFolder has its own manager view (default parent view, such as **History Import Manager**).

driver-ConfigRule

ConfigRule is used to determine the configuration overrides for histories when they are exported to the station. By default, the parent container HistoryNetworkExt (History Policies) contains only a single ConfigRule, named "Default Rule." Default settings are to configure "all" histories exported to the station as "unlimited" capacity.

For more details, see "Config Rules" on page 2-7 and "About History Policies" on page 2-7.

driver-DelimitedFileImport

DelimitedFileImport is a history file import descriptor used to create a Niagara history based upon data in a (local) delimited text file, such as comma-separated-values (CSV) or tab-delimited values (delimiter to use is configurable). These import descriptors reside under the HistoryNetworkExt (**`Histories`** extension) of a FileDevice in a FileNetwork. You use the Delimited File Import Manager view of the Histories extension to add history file import descriptors.

Note: *This import descriptor is similar to the ExcelCsvFileImport descriptor, but uses a "dumb" string tokenizer to parse each line of the specified file, and separates table columns based solely on the specified delimiter. Only "non-complex" CSV files should be imported using it, or any "non-CSV" delimited file (such as tab-delimited, for example). For any CSV file created by Microsoft Excel, use the ExcelCsvFileImport 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" on page 1-46. For other configuration properties, see "Properties of history file import descriptors" on page 5-2.

driver-DriverContainer

DriverContainer is used by convention to store all DeviceNetworks in a station database. The DriverManager is its primary view. See "About the Driver Manager" on page 1-4.

driver-ExcelCsvFileImport

ExcelCsvFileImport is a history file import descriptor used to create a Niagara 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" on page 1-46. 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:

- **Value Facets**
  Lets you specify the units with which to display values imported from the delimited file. On the import descriptor's property sheet, this is property is found under "Config Overrides".
- **Time Zone**
  Lets you specify the time zone for the imported history. On the import descriptor's property sheet, this is property is found under "Config Overrides".
- **File**
  (*Important*) Identifies the local delimited file to import, using standard "ord" file syntax. Typically, you simply click the folder icon to open the **`File Chooser`**, and navigate as needed to click-select an *absolute* file ord path to the file.
  Of, 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**
  Default is disabled. If set to enabled, the entire history is re-filled afresh upon each import. If left at default, only new data is appended to the history upon every import.
- **Row Start**
  Along with Row End, lets you specify the start and end rows (lines) in the file to use for importing. Note that both properties are zero-based, meaning that a Row Start of 1 means that it skips the first line (usually column headers), and begins importing on the second line.
- **Row End**
  See Row Start, above. Row End is optional, and defaults to go to the end of the file (none).
- **Delimiter**
  (appears only if a DelimitedFileImport descriptor) Specifies the text character used in the file to separate columns. For any ExcelCsvFileImport descriptor, a comma (",") is used by default.

- **Timestamp Column Index**
  (*Required*, zero-based) Left-to-right index of the column in the file used to import the timestamp. For example: if first column, this value is “`0`”; if second column, this value is “`1`”; and so on.
- **Timestamp Format**
  (*Required*) String that specifies how timestamps are formatted in the file. A drop-down control lets you select among several common formats—after selection, you can further edit if necessary (may be necessary if you execute, and the descriptor is in fault). For timestamp format details and examples, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html
- **Value Column Index**
  (*Required*, zero-based) Left-to-right index of the column in the file used to import the value. For example: if the second column, this value is “`1`”; if the fifth column, this value is “4”; and so on.
- **Value Format**
  Specifies the value type, meaning the type of history record to create, as one of the following:
  - Numeric Type — (default) for a numeric type history.
  - String Type — for a string type history.
  - Boolean Type — for a boolean (two-state) type history.
- **Status Column Index**
  (Usage optional). The left-to-right index (zero-based) of the column in the file used to import the status, if available. Note that the imported status *value* must match the Niagara “status bits” implementation in encoded-integer format, using the following decimal equivalent values (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`
- **Identifier Column Index**
  (Usage optional). The left-to-right index (zero-based) of the column in the file used to filter rows for inclusion, in combination with the Identifier Pattern property value (below). Default value is None (no row filtering).
- **Identifier Pattern**
  (Usage optional). Specifies the text string in the Identifier Column (above) that is searched for in all rows, where any row with the matching text string is imported. Note that wildcard (“`*`”) characters are supported. Default if value is `*` (everything matched).

driver-FileDevice

FileDevice is the “device level” container to import *local* delimited-type files (such as CSV) as Niagara histories. It resides under the FileNetwork. It has common device properties (see “Device status properties” on page 1-22) 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 drop-down control beside folder icon, choose **`Directory Chooser`**). This allows you to enter *relative* file ord values in the “File” property of history file import descriptors (under its child **`Histories`** extension). Otherwise (using the default “null” Base Ord), you specify absolute “File” ord values in child descriptors. For related details, see “Properties of history file import descriptors”.

Note: *Typically, a FileNetwork is configured with only a single FileDevice, with a default “null” value Base Ord property. However, if you wish 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.*

Note: *Unlike many fieldbus drivers, there is no “frozen” LocalFileDevice in a FileNetwork—any FileDevice named “LocalFileDevice” is simply a standard FileDevice component.*

driver-FileHistoryDeviceExt

FileHistoryDeviceExt is the `driver` module implementation of HistoryDeviceExt, and is the only device extension under a FileDevice. The default view is the DelimitedFileImportManager view, which you use to add new (or edit existing) file import descriptors. Each descriptor adds a Niagara history in the local history space.

driver-FileHistoryWorker

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

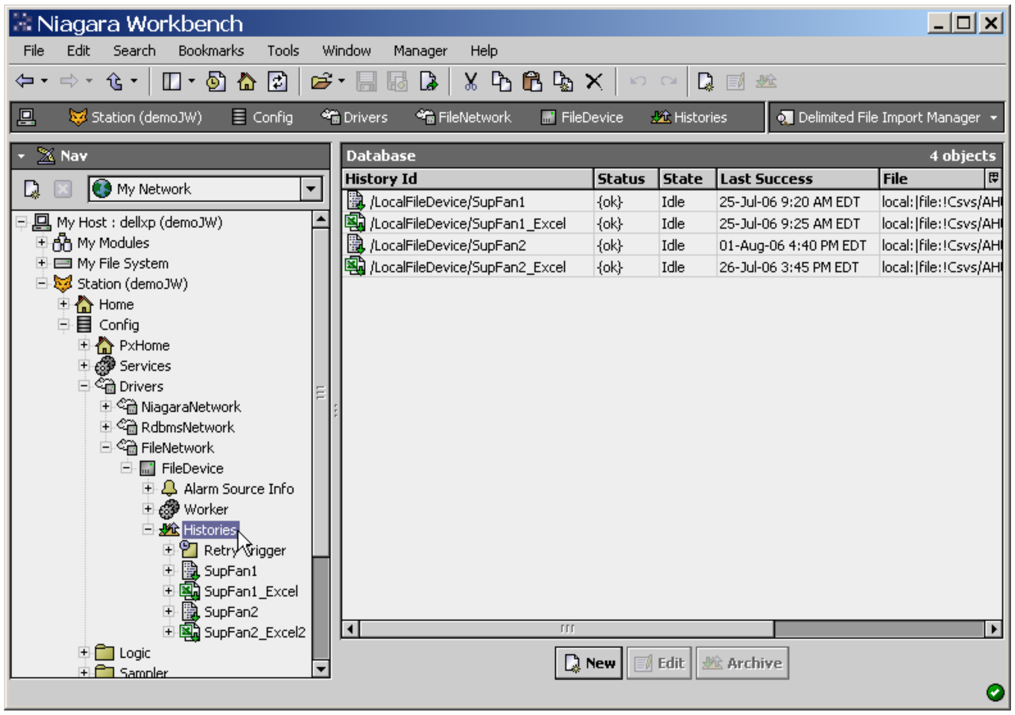
driver-FileNetwork

FileNetwork is available in the driver module, and acts as the “network level” container for one or more FileDevice components. The FileDeviceManager is its primary view.

The purpose of a FileNetwork is to import data from *local* delimited-type files (such as CSV), as Niagara histories. Often, a FileNetwork contains only a single FileDevice. Unlike in true “field bus” networks, the standard NiagaraAX driver architecture (network: device: device extensions) provides no “real hardware” equivalency, this is simply a modeling “convention.”

Figure 5-1 shows the architecture of a FileNetwork in the Nav tree, where the active (and most used) view is the DelimitedFileImportManager view of the **`Histories`** extension (FileHistoryDeviceExt) of the FileDevice.

Figure 5-1 *FileNetwork architecture uses standard “device” modeling*



Database 4 objects

History Id	Status	State	Last Success	File
/LocalFileDevice/SupFan1	{ok}	Idle	25-Jul-06 9:20 AM EDT	local:|file:!Csvs/AH
/LocalFileDevice/SupFan1_Excel	{ok}	Idle	25-Jul-06 9:25 AM EDT	local:|file:!Csvs/AH
/LocalFileDevice/SupFan2	{ok}	Idle	01-Aug-06 4:40 PM EDT	local:|file:!Csvs/AH
/LocalFileDevice/SupFan2_Excel	{ok}	Idle	26-Jul-06 3:45 PM EDT	local:|file:!Csvs/AH

Note that a FileDevice has *none* of the other “standard” device extensions (Points, **`Schedules`**, or **`Alarms`**). See “Usage notes” for more details on a File Network.

Usage notes The following notes apply to using a FileNetwork.

- The FileNetwork requires special licensing in the host’s Tridium license—requiring 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). Without proper licensing, the FileNetwork and child components will have a `fault` status, and files will not import as Niagara histories.
- 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. Note this means if a delimited file has two columns: one for date and another for time, you must perform external “upstream” processing (outside of Niagara) to combine into a single column, before importing.
  - It must have a *single* column that contains the “value” required in the imported history. Note that 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 Niagara “status bits” values. See “Properties of history file import descriptors” on page 5-2 for related details.
    Note the status import feature may be used mainly with data that first originated from a Niagara history (that was exported to CSV), then subsequently “manipulated” outside of Niagara.

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

driver-HistoryNetworkExt

HistoryNetworkExt (History Policies) provides two network-level functions for histories, as follows:

- For history *imports*, it contains an "On Demand Poll Scheduler", which has configurable poll rates used in "on demand polling" of imported histories, if so configured.
- For history *exports*, it acts a container for one or more "config rules", which specify how the configuration of the local history (archive) is set when the history is "pushed" (exported) into the station. Configuration rules are applied only when the history is created, that is, when first archived to the station. Changing a rule has no effect on existing histories.

For more details, see "About History Policies" on page 2-7.

driver-HistoryPollScheduler

HistoryPollScheduler (`On Demand Poll Scheduler`) contains properties that specify the available network-level polling rates for imported histories, used in "on demand" polling of an imported history. Such polling can be triggered when the "Live Updates" button in a history chart or history table view is toggled active. For more details, see "Default history policies" on page 2-7.

driver-PingMonitor

The PingMonitor periodically calls "ping" on all the "pingables" to monitor network and device health. PingMonitor provides built-in support to generate alarms when pingables are down. The PingMonitor is available in the property sheet of most networks as "Monitor." See "About Monitor" on page 1-7 for more details.

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

Contains a collection of properties typically used in the driver network's evaluation of both *write requests* (e.g. to writable proxy points) as well as 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 TuningPolicies under a driver's TuningPolicyMap. You can then assign one or more proxy points to a specific TuningPolicy. See"About Tuning Policies" on page 1-8 for more details.

driver-TuningPolicyMap

Container for one or more TuningPolicy(ies), found in the property sheet of most network components. You can create multiple TuningPolicies under a network's TuningPolicyMap. You can then assign one or more proxy points to a specific TuningPolicy. See"About Tuning Policies" on page 1-8 for more details.

Components in fox module

- FoxClientConnection
- FoxFileService
- FoxServerConnection
- FoxService
- FoxSession

fox-FoxClientConnection

FoxClientConnection (`Client Connection`) encapsulates a FoxSession connection initiated by this VM. It is used as the client connection in a station VM by a NiagaraStation, and in Workbench by FoxSession. In the `Client Connection` container under a `NiagaraStation`, you must setup the properties **`Address`**, **`Port`**, **`Username`**, and **`Password`** in order to access a remote station.

Actions You can manually force/test a connection using **`Manual Connect`** and **`Manual Disconnect`** actions.

fox-FoxFileSpace

FoxFileSpace maps "File:" remotely.

fox-FoxServerConnection

FoxServerConnection encapsulates a 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.

Actions Available Action is **`Force Disconnect`**.

fox-FoxService

FoxService is the Baja component wrapper for the FoxServer daemon. It is commonly used within a NiagaraNetwork, but can be used stand alone to provide basic Fox accessibility.

fox-FoxSession

FoxSession represents a Workbench (client) session to a station running on a NiagaraAX host (server), using a Fox connection on a particular port—in other words, a *station* connection. You see this as an expandable icon in Workbench's Nav tree, following by the station's (*name*), for any station opened in Workbench.

Starting in AX-3.7, a secure Fox station connection (SSL or TLS) is also possible from Workbench, in which case the station connection icon shows a small padlock, like the above.

Also starting in AX-3.7, 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.

fox-ServerConnections

ServerConnections is the container slot to store the current server-side FoxServerConnections from non-station clients. The primary view is the ServerConnectionsSummary. The ServerConnections is available in the fox module.

Components in niagaraDriver module

- BogProvider (Sys Def Provider)
- CyclicThreadPoolWorker
- LocalSysDefStation
- NiagaraAlarmDeviceExt
- NiagaraFileDeviceExt
- NiagaraFileImport
- NiagaraFoxService
- NiagaraHistoryDeviceExt
- NiagaraHistoryExport
- NiagaraHistoryImport
- NiagaraNetwork
- NiagaraPointDeviceExt
- NiagaraPointFolder
- NiagaraProxyExt
- NiagaraScheduleDeviceExt
- NiagaraScheduleImportExt
- NiagaraStation
- NiagaraStationFolder
- NiagaraSysDefDeviceExt
- NiagaraSystemHistoryExport
- NiagaraSystemHistoryImport
- NiagaraTuningPolicy
- NiagaraTuningPolicyMap
- NiagaraUserDeviceExt
- NiagaraVirtualDeviceExt
- ProviderStation

- RoleManager
- SyncTask

niagaraDriver-BogProvider

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

niagaraDriver-CyclicThreadPoolWorker

The CyclicThreadPoolWorker (**Workers**) slot of a 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. See “NiagaraNetwork component notes” on page 2-1 for further details.

niagaraDriver-LocalSysDefStation

LocalSysDefStation (“**Local Station**”) is child container property in a NiagaraStation. It reflects common “Sys Def” properties for the local station, which would be “sync’ed up” to a remote Supervisor station (if the local station was defined as its subordinate). Sys Def is chiefly of interest to developers extending the API.

niagaraDriver-NiagaraAlarmDeviceExt

A NiagaraStation’s NiagaraAlarmDeviceExt (Alarms extension) specifies how alarms from that station are mapped into the current station’s own alarm subsystem, plus provide status properties related to alarm sharing. For more details, see “About the Alarms extension” on page 1-35.

niagaraDriver-NiagaraFileDeviceExt

A NiagaraFileDeviceExt (“**Files**”) is included among the collection of device extensions under any NiagaraStation component. This device extension allows the station to import files, and folders of files, from the associated remote station. The default view is the Niagara File Manager, in which you create NiagaraFileImport descriptors, to discover, add, and edit parameters specifying which files and folders are to be imported. For more details, “About the Files extension” on page 2-54.

niagaraDriver-NiagaraFileImport

NiagaraFileImport is an “import descriptor” that specifies a file or folder of files (including all subfolders and files) that are to be imported to the local station, sourced from the remote parent NiagaraStation. Included are properties for execution time and overwrite policies. For more details, “About Niagara FileImport properties” on page 2-56.

niagaraDriver-NiagaraFoxService

NiagaraFoxService (Fox Service) is a container slot of a NiagaraNetwork, and is a specialization of FoxService that knows how to map server connections to the NiagaraStation serverConnection slot. NiagaraFoxService typically includes ServerConnections. For more details, see “About the Fox Service” on page 2-2.

niagaraDriver-NiagaraHistoryDeviceExt

NiagaraHistoryDeviceExt is the Niagara implementation of HistoryDeviceExt. For more details, see “About the Histories extension” on page 1-34 and “About Histories extension views” on page 1-46.

niagaraDriver-NiagaraHistoryExport

NiagaraHistoryExport defines the export parameters for a local Niagara history, including collection (“push”) times, current status, and remote history Id. NiagaraHistoryExports reside under the History extension of a NiagaraStation in the NiagaraNetwork. For more details, see “Niagara History Export properties” on page 2-35.

niagaraDriver-NiagaraHistoryImport

NiagaraHistoryImport defines the local import parameters for a remote Niagara history, including collection (“pull”) times, current status, local history Id, and config overrides. NiagaraHistoryImports reside under the History extension of a NiagaraStation in the NiagaraNetwork. For more details, see “Niagara History Import properties” on page 2-31.

niagaraDriver-NiagaraNetwork

NiagaraNetwork models NiagaraAX devices (NiagaraStations) that the current station can communicate with. The primary view is the StationManager. The NiagaraNetwork includes the NiagaraFoxService. For more details, see “About the Niagara Network” on page 2-1.

niagaraDriver-NiagaraPointDeviceExt

NiagaraPointDeviceExt is the Niagara implementation of PointDeviceExt. The primary view is the PointManager. For general information, see "About the Points extension" on page 1-27, and for specific details see "About the Bql Query Builder" on page 2-21 and "Niagara proxy point notes" on page 2-24.

niagaraDriver-NiagaraPointFolder

NiagaraPointFolder is the Niagara implementation of a folder under a NiagaraStation's Points extension. You add such folders using the New Folder button in the PointManager view of the Points extension. Each NiagaraPointFolder has its own PointManager view.

niagaraDriver-NiagaraProxyExt

NiagaraProxyExt is the Niagara implementation of BProxyExt. For more details see "About proxy points" on page 1-27 and "Niagara proxy point notes" on page 2-24.

niagaraDriver-NiagaraScheduleDeviceExt

NiagaraScheduleDeviceExt is the Niagara implementation of a Schedules device extension. For general information, see "About the Schedules extension" on page 1-36, and for more specific details see "Station Schedules import/export notes" on page 2-28.

niagaraDriver-NiagaraScheduleImportExt

NiagaraScheduleImport defines the local import parameters for a remote Niagara schedule. For more details, see "Station Schedules import/export notes" on page 2-28.

niagaraDriver-NiagaraStation

NiagaraStation models a platform running a Niagara station via a "regular" (unsecured) Fox connection, including a a JACE, SoftJACE, or Supervisor.

NiagaraStation (with padlock on icon) models a platform running a Niagara station via a secure (SSL or TLS) *Foxs* connection, including a including a JACE, SoftJACE, or Supervisor. Applies only if the remote host is running AX-3.7 or later, is licensed for SSL (feature "`crypto`"), and is configured for Foxs.

In either case, the NiagaraStation name must map to Station.stationName. For more details, see "NiagaraStation component notes" on page 2-14.

For details related to a Foxs connection, see "Discovery notes when stations use secure Fox (Foxs)" on page 2-10, and also "About the Fox Service" on page 2-2.

Ping Perform a ping test of the device.

niagaraDriver-NiagaraStationFolder

NiagaraStationFolder is the Niagara implementation of a folder under a NiagaraNetwork. You add such folders using the New Folder button in the StationManager view of the Niagara network. Each NiagaraStationFolder has its own StationManager view. Station folders can be useful in very large systems to organize NiagaraStation components.

niagaraDriver-NiagaraSysDefDeviceExt

NiagaraSysDefDeviceExt (**`Sys Def`**) is included among the collection of device extensions under any NiagaraStation component, and is chiefly of interest to developers. This Sys Def extension has two child container properties, "RoleManager" and "SyncTask", each with a collection of related properties. Together with the NiagaraNetwork's "Sys Def Provider" (BogProvider) component, the API helps define the organization of stations in a NiagaraNetwork in a known hierarchy, and allows synchronization of basic information up that hierarchy.

For more details, "About Sys Def components" on page 2-51.

niagaraDriver-NiagaraSystemHistoryExport

NiagaraSystemHistoryExport defines the "system tags" text patterns used to export local Niagara histories into the target NiagaraStation, in addition to other export parameters including collection ("push") times and current status. Along with NiagaraHistoryExport descriptors, NiagaraSystemHistoryExport descriptors reside under the History extension of a NiagaraStation in the NiagaraNetwork. However, SystemHistoryExport descriptors utilize "System Tags" properties of local history extensions, instead of unique history IDs.

For more details, see "Niagara History Export properties" on page 2-35 and "Using System Tags to export Niagara histories" **on page 2-37**.

niagaraDriver-NiagaraSystemHistoryImport

NiagaraSystemHistoryImport defines the "system tags" text patterns used to import remote Niagara histories from the target NiagaraStation, in addition to local import parameters, including collection ("pull") times, current status, local history Id, and config overrides. Along with NiagaraHistoryImport descriptors, NiagaraSystem HistoryImport descriptors reside under the History extension of a NiagaraStation in the NiagaraNetwork. However, SystemHistoryImport descriptors utilize "System Tags" properties of remote history extensions, instead of unique history IDs.

For more details, see "Niagara History Import properties" on page 2-31 and "Using System Tags to import Niagara histories" on page 2-32.

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 Niagara 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. See"About Tuning Policies" on page 1-8 for general information, and "Niagara Tuning Policy notes" on page 2-2 for specific details.

niagaraDriver-NiagaraTuningPolicyMap

Container for one or more NiagaraTuningPolicy(ies), found in the NiagaraNetwork's property sheet. If needed, you can create multiple tuning policies. You can then assign one or more Niagara proxy points to a specific policy.

niagaraDriver-NiagaraUserDeviceExt

The NiagaraUserDeviceExt (**Users**) container of a NiagaraStation holds properties that enable/configure network user synchronization "in" and "out" of this station, in relation to the station with its NiagaraNetwork. There is no special view (apart from property sheet) on this Users device extension, nor is it a container for other components.

For more details, see "About the Users extension" on page 2-15.

niagaraDriver-NiagaraVirtualDeviceExt

The NiagaraVirtualDeviceExt (**Virtual**) is the Niagara driver implementation of the Baja virtual gateway in a AX-3.7 or later 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." Note other object spaces are `Files` and `History`. For a general explanation about Baja virtual components, refer to "About virtual component spaces" on page 1-24.

For details on the Niagara Virtual Gateway, see "About the Niagara Virtual Device Ext" on page 2-40.

niagaraDriver-ProviderStation

ProviderStation ("*stationName*") is a child container under the BogProvider ("Sys Def Provider") container in a 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-RoleManager

RoleManager is child container under a 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, introduced in AX-3.5, is chiefly of interest to developers extending the API.

niagaraDriver-SyncTask

SyncTask is child container under a 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, introduced in AX-3.5, is chiefly of interest to developers extending the API.

Components in niagaraVirtual module

- NiagaraVirtualBooleanPoint
- NiagaraVirtualBooleanPoint
- NiagaraVirtualCache
- NiagaraVirtualCache
- NiagaraVirtualComponent
- NiagaraVirtualDeviceExt

- NiagaraVirtualEnumPoint
- NiagaraVirtualEnumWritable
- NiagaraVirtualNumericPoint
- NiagaraVirtualNumericWritable
- NiagaraVirtualStringPoint
- NiagaraVirtualStringWritable

niagaraVirtual-NiagaraVirtualBooleanPoint

NiagaraVirtualBooleanPoint is the Niagara driver implementation of a virtual BooleanPoint. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualBooleanWritable

NiagaraVirtualBooleanWritable is the Niagara driver implementation of a virtual BooleanWritable. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualCache

NiagaraVirtualCache (Cache) is the component representation of a NiagaraNetwork's virtual component cache. It provides a default **`Niagara Virtual Cache View`** to examine persisted cached data (in file or files specified in the virtual Cache Policy). For more details, see "Niagara virtuals cache (Virtual Policies)" on page 2-42.

niagaraVirtual-NiagaraVirtualCachePolicy

NiagaraVirtualCachePolicy (Cache Policy) is the container for a NiagaraNetwork's cache of virtual components, holding configuration properties as well as a component representing the Cache. For more details, see "Niagara virtuals cache (Virtual Policies)" on page 2-42.

niagaraVirtual-NiagaraVirtualComponent

NiagaraVirtualComponents (or simply Niagara "virtuals") are the Niagara driver implementation of Baja Virtual components. They reside under the NiagaraVirtualDeviceExt ("Virtual" slot) of each NiagaraStation. Typically, note that Niagara virtuals are possible only in an Supervisor station's NiagaraNetwork (the niagaraDriver feature in the host's license must have "virtual" attribute set to true). In this scenario, if the "Virtuals Enabled" property of a NiagaraStation is set to true, the entire component structure of that station can be dynamically modeled using Niagara virtuals.

Note that the Workbench icon for each Niagara virtual has a small "ghost" ( ) superimposed in the lower right over the "normal" icon for that type—a visual reminder that you are looking into the "virtual component space" that represents that station.

For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualDeviceExt

The NiagaraVirtualDeviceExt (**`Virtual`**) is the Niagara driver implementation of the Baja virtual gateway in a AX-3.7 or later 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." Note other object spaces are `Files` and `History`. For a general explanation about Baja virtual components, refer to "About virtual component spaces" on page 1-24.

For details on this Niagara Virtual gateway, see "About the Niagara Virtual Device Ext" on page 2-40.

niagaraVirtual-NiagaraVirtualEnumPoint

NiagaraVirtualEnumPoint is the Niagara driver implementation of a virtual EnumPoint. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualEnumWritable

NiagaraVirtualEnumWritable is the Niagara driver implementation of a virtual EnumWritable. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualNumericPoint

NiagaraVirtualNumericPoint is the Niagara driver implementation of a virtual NumericPoint. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualNumericWritable

NiagaraVirtualNumericWritable is the Niagara driver implementation of a virtual NumericWritable. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualStringPoint

NiagaraVirtualStringPoint is the Niagara driver implementation of a virtual StringPoint. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

niagaraVirtual-NiagaraVirtualStringWritable

NiagaraVirtualStringWritable is the Niagara driver implementation of a virtual StringWritable. They are among the eight type-specific "Niagara virtual" components. For more details, see "About Niagara virtual components" on page 2-38.

Components in serial module

- SerialHelper

serial-SerialHelper

Container slot that handles the serial port configuration of any serial-based driver (found under the driver's Network-level component as "Serial Port Config").

Note: *Also found under any SerialTunnel (in station's TunnelService) to support a tunneling connection to the same type of network. In this case, configure it the same as the SerialHelper of the associated driver network.*

SerialHelper contains the following properties:

- **Status**
  Read-only status of the serial port, where it can be `fault` if a problem was detected, `down` if the serial port is not properly configured, or `ok` otherwise.
- **Port Name**
  String for the serial port, as known to the host platform. For example, `COM1` or `COM4`.
- **Baud Rate**
  Baud rate used in serial communications. Select from enumerated drop-down selections, from `Baud50` through `Baud115200`. Default value is `Baud9600`.
- **Data Bits**
  Data bits used in serial communications. Select from enumerated drop-down selections, from `Data Bits8` through `Data Bits5`. Default value is `Data Bits8`.
- **Stop Bits**
  Number of stop bits used, as either `Stop Bit1` (default) or `Stop Bit2`.
- **Parity**
  Parity to use, as either `None`, `Odd`, `Even`, `Mark`, or `Space`. Default value is `None`.
- **Flow Control Mode**
  Flow control to use, as either `RtsCtsOnInput`, `RtsCtsOnOutput`, `XonXoffOnInput`, `XonXoffOnOutput`, or *none* (all checkboxes cleared). Default value is none.

Components in tunnel module

- TunnelService
- SerialTunnel
- TunnelConnection

tunnel-TunnelService

Station server for "application tunneling," where remote PCs with a NiagaraAX 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 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.

The TunnelService contains the following properties:

- **Enabled**
  Boolean that must be `true` to support any tunnel server operations.
- **Server Port**
  Software port monitored for incoming client tunnel connections. Default port is 9973.
- **Status**
  TunnelService status, which should be `ok` (no special licensing required).
- **Connections**
  Number of active tunnel connections, which ranges from 0 (no active) to the number of child tunnel components.

tunnel-SerialTunnel

. SerialTunnel 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 JACE controller.

You can add one or more SerialTunnels under a station's TunnelService. Each SerialTunnel associates with one specific driver network (and corresponding JACE serial port). See "Serial tunneling" on page 3-2 for more details.

Properties of the SerialTunnel are described as follows:

- **Enabled**
  Boolean slot that must be enabled (true) to permit tunneling.
- **Connections**
  Read-only slot to indicate the number of tunnel connections, as either 0 (none) or 1 (maximum).
- **Status**
  Read-only status of the serial tunnel, typically `ok` (unless `fault` for no supporting COM port).
- **Identifier**
  Read-only "reflection" of the entered Port Name slot in the Serial Port Config container (below), used as the "Tunnel Name" when configuring the client-side Serial Tunneling dialog.
- **Serial Port Config (container)**
  Holds configuration of the JACE serial port as used by the specific driver network. See SerialHelper on page 11 for slot descriptions.

In addition, a SerialTunnel has an available (right-click) *action*:

- **Disconnect All**
  Disconnects any active connection through this SerialTunnel (maximum of 1), causing removal of the "TunnelConnection" below it. On the remote (serial tunnel client) side, a popup message "Connection closed by remote host" is seen.
  Note: *Any TunnelConnection component also has its own "Disconnect" action, which effectively performs the same function.*

tunnel-TunnelConnection

. TunnelConnection is a dynamically-added component under a tunnel component (such as a SerialTunnel) that reflects read-only information about this current tunnel connection.

Properties of a TunnelConnection (any tunnel type) are as follows:

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

In addition, a TunnelConnection has an available (right-click) *action*:

- **Disconnect**
  Disconnects the active tunnel connection, removing the parent TunnelConnection component. This causes a popup “Connection closed by remote host” to be seen on the client tunnel side.
  Note: *A SerialTunnel component also has its own “Disconnect All” action, which effectively performs the same function.*