

Technical Document

Niagara Engineering Notes

niagara

About this Guide

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

Product Documentation

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

Document Content

This document serves as a collection of Niagara topics that may be helpful to both Systems Integrators and Engineers.

Topics in this document should be relevant to many users of Niagara 4 even though most were written initially for NiagaraAX

Each individual “Engineering Notes” topic is included as a separate chapter in the print or PDF rendering of this Guide.

- [Document change log](#)
 - [Related documentation](#)
- Related information is available in the following documents.

Document change log

March 7, 2022

Updated properties table in the Tokens component chapter.

June 25, 2020

Corrected bql code example in the topic, “Weekly Electrical Demand Report”.

March 8, 2017

Made the following changes in the chapter, Using the dashboard feature. Corrected steps in the topic, “Setting up a dashboard.” Minor changes to the topics, “Editing a dashboard” and “Dashboard commands.” Significant changes to several answers in the topic, “Frequently asked questions.”

January 27, 2017

Added chapter on Using Dashboards which describes the Niagara 4 Dashboards feature including: setting up and editing dashboards as well as the Dashboard Service and related components. Also, fixed cross-reference hyperlinks within the chapter, bacnetUtil Component Usage.

October 7, 2016

JACE Hardware Scan Service engineering notes to include the JACE-8000 controller.

December 14, 2015

Appended notes on BACnet in N4 and use of bacnetUtil components.

August 19, 2015

Initial release publication.

Parent topic: [About this Guide](#)

Related documentation

Related information is available in the following documents.

- *Getting Started with Niagara*
- *Niagara Platform Guide*
- *Niagara Developer Guide*
- *Niagara Web Charts Guide*

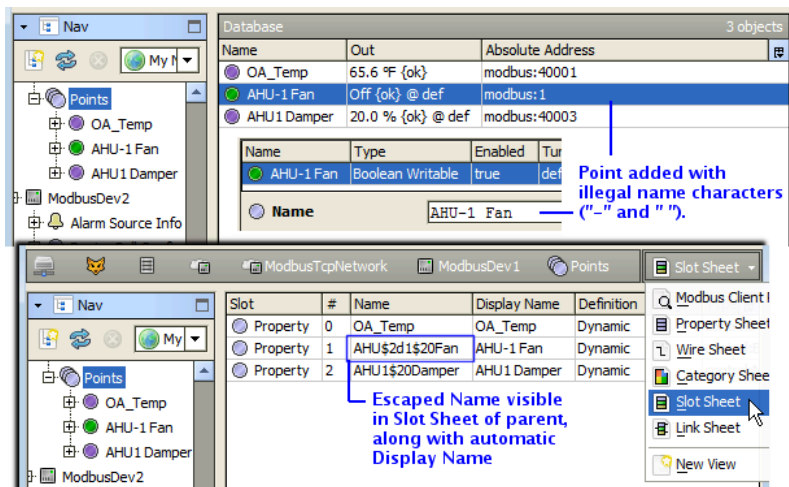
Parent topic: [About this Guide](#)

Display Names

Display names provide an optional method to label components, slots, and histories in a station to increase human readability.

For a component, a display name differs from a regular *name*, in that a display name follows certain rules, and is included as a portion of the *ord* for the component. Component name rules allow only alphanumeric characters (a-z, A-Z, 0-9), and underscores (_), where name must begin with an alpha (a-z, A-Z) character.

Figure 1. Component created with illegal character(s) in name has an escaped name

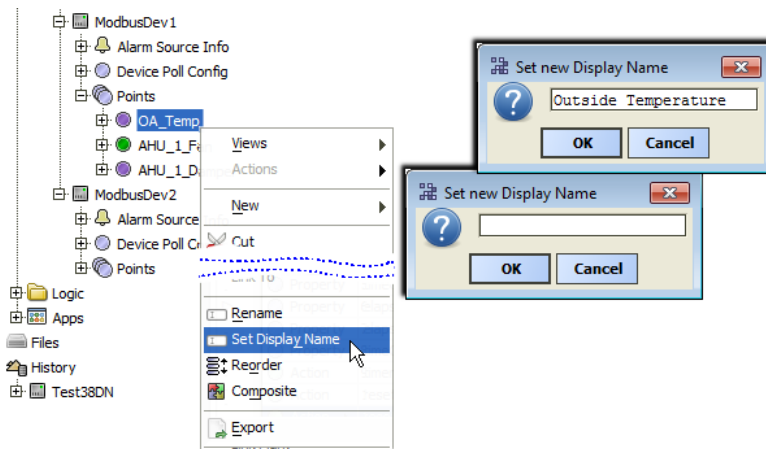


If you add a component with a name that breaks the rules, for example, a name with a space character, hyphen, or name that begins with a numeral, the component's name is created with "escaped characters", and your typed entry is used as the display name. As shown in the preceding figure, the escaped name is visible on the *slot sheet* of the component's *parent*. Each escaped character begins with a dollar sign (\$) followed by the hexadecimal ASCII code for that character, for example "\$2d" for a hyphen, or "\$20" for a space character.

Note: Using illegal characters when naming components is considered a poor practice, as escaped names make ords longer and more obscure. Such ords can also cause confusion in other areas of the system. You should assign component names using "CamelCase" and/or underscore(s) instead of spaces or other punctuation— as shown above for OA_Temp. Then as needed, explicitly set a display name for components.

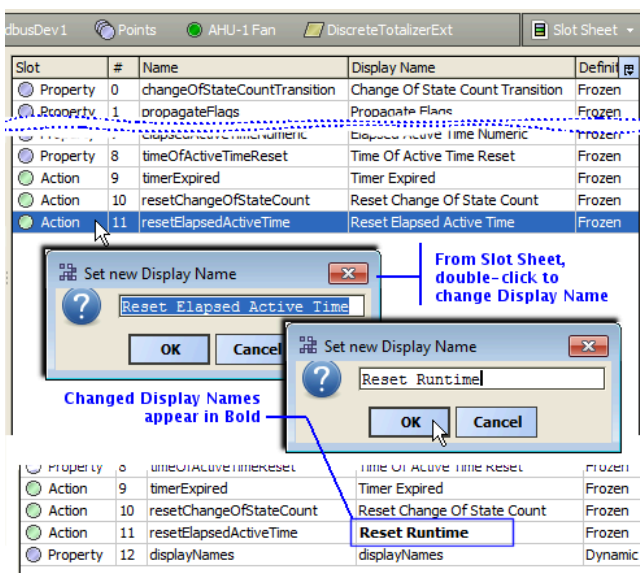
As shown in the preceding figure, you can do this from the Workbench Nav tree, by right-clicking the component for the **Set Display Name** command.

Figure 2. Example edit of a component to set a display name



Slots of many components such as control points include frozen properties and actions, where each has a default name. If needed, go to the slot sheet of the component and edit these names, resulting in explicit display names. This is commonly done to customize names of point actions.

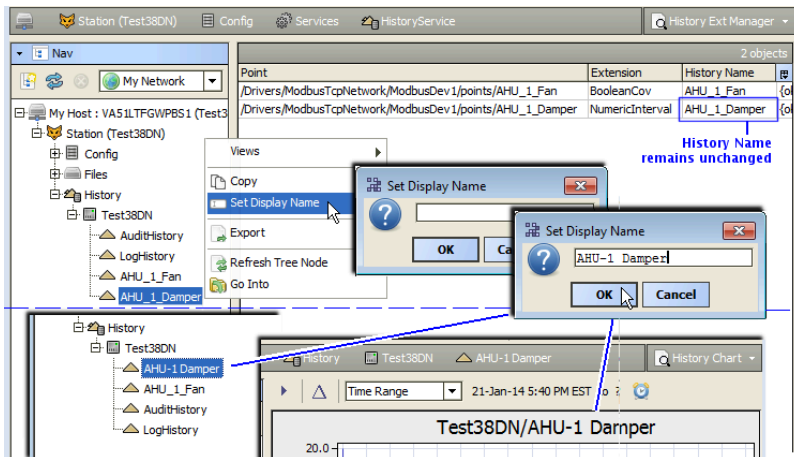
Figure 3. Example edit of an action slot on DiscreteTotalizerExt



For example, instead of Reset Elapsed Active Time as an action for a DiscreteTotalizerExt of a BooleanWritable point, you might edit it to Reset Runtime, as shown in the preceding figure.

Histories also support display names. In the Workbench Nav tree, right-click a history for the **Set Display Name** command.

Figure 4. Example edit of history to assign a display name (right-click history)



If you assign a display name to a history, its original history ID is not affected. The display name is simply used when displaying the history in various views into the system.

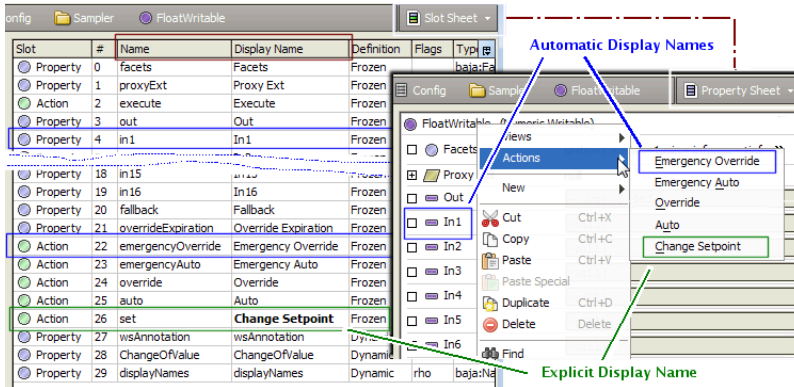
- [Automatic display names](#)
Niagara 4 automatically renders display names for most frozen slots, without requiring a `displayNames` slot on the parent component. You can see this from the slot sheet of a parent component.
- [Display name storage](#)
Display names of components and slots are stored on the *parent* of a component or slot.
- [Manually setting display names](#)
This section provides instructions for manually setting display names for individual components and slots while working in Workbench.
- [Using the Batch Editor for display names](#)
The **Batch Editor** view of a station's ProgramService provides a way for you to edit or add explicit display names in a *batch* process.

Automatic display names

Niagara 4 automatically renders display names for most frozen slots, without requiring a `displayNames` slot on the parent component. You can see this from the slot sheet of a parent component.

When you look at the slot sheet of a component, note that slots for most *frozen* properties and actions, as well as any frozen child components, use a similar *naming pattern*. This pattern begins with a lowercase letter and uses “camelCase” (no spaces). If the name includes multiple words, a capital letter begins each new word.

Figure 1. Automatic display names for frozen slots vs. explicit display names



For example, as shown in the preceding figure of the slot sheet of a writable control point, see slot names `in1` — `in16` for properties and `emergencyOverride` and `override` for actions. Display names appear for these on property sheets and action menus, such as **In1**, **In16**, **Emergency Override**, and **Override**.

Note in this example, the `set` action has been given an *explicit* display name: Change Setpoint. Looking at the slot sheet, it is apparent this is the *only* slot with an explicit display name, as it is the only display name in **bold text**.

Note: If using the ProgramService **Batch Editor** to edit or add display names, it is important to enter the BNameMap “key” values using the actual *names* for slots, and not the automatic display names. For example, enter “override” versus “Override”, or “auto” versus “Auto”.

Parent topic: [Display Names](#)

Display name storage

Display names of components and slots are stored on the *parent* of a component or slot.

Explicitly-assigned display names are stored in the display “name map” of the *parent* component. This applies whether you manually assigned a display name, or used the Batch Editor for adding/editing display names.

displayNames slot

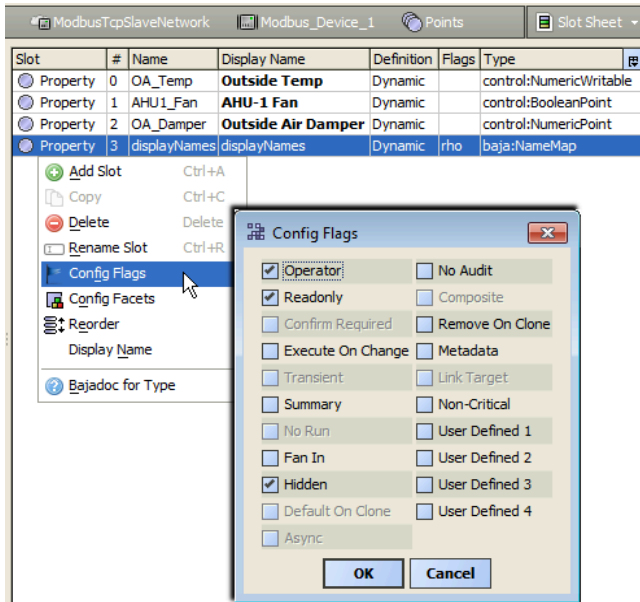
The “displayNames” slot on the parent component uses the data type `baja:NameMap`. This slot is created upon the first explicit display name given to a child slot. If the `displayNames` slot already exists, any new explicit display name is appended to its BNameMap value.

From the slot sheet of a parent, all child slots that have been explicitly-assigned a display name appear with a **bold** Display Name value.

Config flags

Often by default, the `displayNames` slot of a component has config flags set to “rho”, meaning it is read-only, hidden (from property sheet), and operator-level.

Figure 1. Default config flags for displayNames slot of a component or container

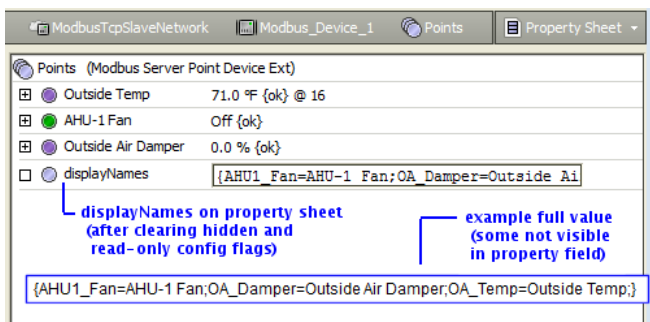


When using the **Batch Editor** to edit display names, you often need to *change* config flags on `displayNames` slots. You can do this either from slot sheets of components, or in a batch method via the Batch Editor.

NameMap value

After “unhiding” a `displayNames` slot from the slot sheet, you can see its NameMap value on the property sheet—or at least a portion of this value (the property field size is fixed, so typically some of the NameMap value is obscured).

Figure 2. Example “Name Map” value of `displayNames` slot of parent with 3 entries



In the preceding figure, the property sheet shows a portion of the NameMap value in the unhidden `displayNames` slot.

BNameMap syntax

The Baja NameMap value of the `displayNames` slot uses a “key=value” pair syntax, as follows:

```
{slotFirst=displayValue1;slotNext=displayValue2;slotLast=displayValue3;}
```

If manually-assigning display names, this syntax is unimportant—as it is always incorporated. However, if using

the **Batch Editor** to edit or add `displayNames` slots, then you must follow this syntax exactly. Note even if only one child slot is assigned a display name, a semi-colon is required at the end of the value string. For example:

```
{slot=displayValue;}  
{OA_Temp=Outside Temp;}
```

Note if the NameMap value contains multiple key-value pairs, it makes no difference in the order that they are listed.

Parent topic: [Display Names](#)

Manually setting display names

This section provides instructions for manually setting display names for individual components and slots while working in Workbench.

Typically, it is best to set explicit display names for components before replicating them in a station—otherwise, you may need to repeat this task for each one.

There are two basic ways to manually set explicit display names:

- [Set a component display name](#)
- [Set display names from slot sheets](#)
- [Set a component display name](#)
You can quickly set a display name for a component.
- [Set display names from slot sheets](#)
From the slot sheet of a component you can set explicit display names for its slots, often done for action slots of a control point. And, from the slot sheet of a container component you can conveniently set display names for its child components (slots).

Parent topic: [Display Names](#)

Set a component display name

You can quickly set a display name for a component.

Station opened in Workbench.

Set the display name for a component in the Nav tree or in a property sheet

1. In the Nav tree, expand the station's **Config** node to find the component, or locate the component in a property sheet.
2. Right-click the component, and select **Set Display Name** from the popup menu.
3. In the **Set new Display Name** popup dialog, type the desired name to display for the component, and click **OK**.

The display name is added to the parent container's `displayNames` slot. When the property sheet or Nav tree is refreshed, the component should now display with the name you entered.

Note: Unless the parent container's `displayNames` slot was originally added using the **Batch Editor**, this slot may have config flags set as "rho" (Readonly, Hidden, Operator). If you want to use the **Batch Editor** to make

future changes, you need to clear the “Readonly” config flag from this slot. If needed, you can do this from the slot sheet or by using the Batch Editor.

Parent topic: [Manually setting display names](#)

Set display names from slot sheets

From the slot sheet of a component you can set explicit display names for its slots, often done for action slots of a control point. And, from the slot sheet of a container component you can conveniently set display names for its child components (slots).

Station opened in Workbench.

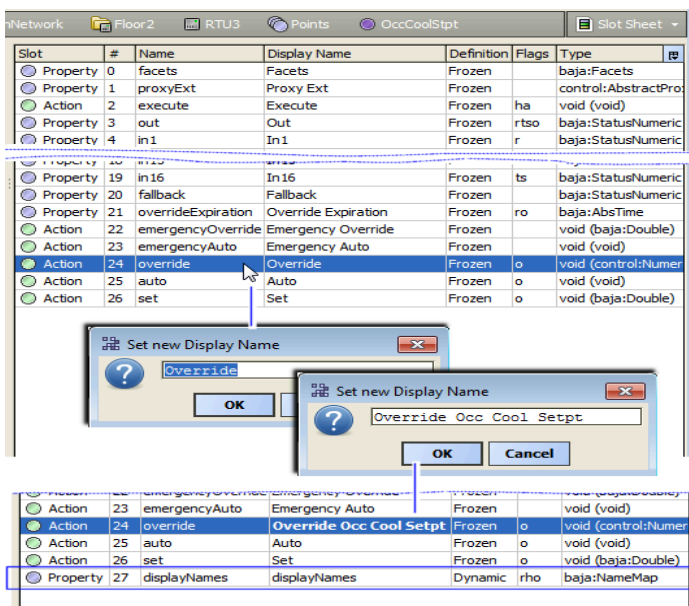
Set display names for slots of a control point or a container from slot sheet

1. In the Nav tree, expand the station’s **Config** node to find the container component or control point, or locate it in a property sheet or wire sheet.
2. Right-click the component, and select **Views > Slot Sheet** from the popup menu.
3. In the slot sheet, double-click a slot to name.
4. In the **Set new Display Name** popup dialog, type the desired name to display for the child component (or slot, such as an action), and click **OK**.
5. Repeat (double-click) other slots to name, as needed.

Each named slot now displays in **bold** on the slot sheet. The display name is added to the NameMap value of the `displayNames` slot. Depending on context, after refreshing the Nav tree or view or accessing the action, the item should now display with the name you entered.

The figure below shows the slot sheet of a control point where the first item (action slot “override”) is being given a display name.

Figure 1. Example first edit of slot display name for a control point slot (action)



As shown in the preceding figure, note the Display Name value now appears in **bold** text, and a new `displayNames` slot has been added to the slot sheet. You can continue to set display names for other slots in

this point, for example one more is shown in the following figure.

Figure 2. Second display name added, as shown in bolded Display Name text

| | | | | | | |
|----------|----|---------------|-------------------------------------|---------|-----|----------------|
| Action | 23 | emergencyAuto | Emergency Auto | Frozen | | void (void) |
| Action | 24 | override | Override Occ Cool Setpt | Frozen | o | void (control: |
| Action | 25 | auto | Release Occ Cool SP Override | Frozen | o | void (void) |
| Action | 26 | set | Set | Frozen | o | void (baja:Do |
| Property | 27 | displayNames | displayNames | Dynamic | rho | baja:NameMa |

As shown in the preceding figure, setting multiple display names does not add additional `displayNames` slots on the component—all the display names are held in the NameMap value of the one `displayNames` slot.

Note:

Unless the `displayNames` slot was originally added using the **Batch Editor**, this slot often has config flags set as “rho” (Readonly, Hidden, Operator). You can still add and re-edit display names from the slot sheet, or on components use the right-click **Set the Display Name** command.

However, note to allow Batch Editor changes to display names, you need to clear the “Readonly” config flag from `displayNames` slots. If needed, do this either working in individual slot sheets, or else in batch mode via the Batch Editor.

Parent topic: [Manually setting display names](#)

Using the Batch Editor for display names

The **Batch Editor** view of a station’s ProgramService provides a way for you to edit or add explicit display names in a *batch* process.

CAUTION: Before using the Batch Editor, always save and backup the station. It is easy to make errors using the Batch Editor, and there is *no undo*. Therefore in a worst-case scenario, you can always reinstall the saved station.

- [Using the Batch Editor to change existing display names](#)
The **Batch Editor** view of the station’s ProgramService allows you to *change* existing display names of slots in control points or even container components.
- [Using the Batch Editor to add display names](#)
The **Batch Editor** view of the station’s ProgramService allows you to *add* display names for slots in control points or even child components of container components.
- [Using the Batch Editor to change slot flags](#)
The **Batch Editor** lets you batch edit *config flags* for the `displayNames` slot in multiple components. Often this is useful when using the Batch Editor to edit or add display names.
- [Troubleshooting batch edited display names](#)
Using the **Batch Editor** for adjusting display names can save time; however, it easy to overlook some things that prevent success. This section lists some common pitfalls and recommendations when using this method.

Parent topic: [Display Names](#)

Using the Batch Editor to change existing display names

The **Batch Editor** view of the station's ProgramService allows you to *change* existing display names of slots in control points or even container components.

The station must be running and opened in Workbench, with the `program` module installed on the Niagara host. The **ProgramService** should be in the station's **Services** folder.

CAUTION: Before using the Batch Editor, always *save and backup the station*. It is easy to make errors using the Batch Editor, and there is *no undo*. Therefore in a worst-case scenario, you can always reinstall the saved station from your backup.

Batch Editor changes to any component's `displayNames` slot are not possible if that slot has the "Readonly" flag set. If necessary, clear the "Readonly" flag from the `displayNames` slot of any component you wish to edit using the Batch Editor. You can do this either from each individual slot sheet, or else by using the Batch Editor.

1. In Workbench with the station opened, access the **Batch Editor** (in the Nav tree, expand the Config, Services node and double-click **ProgramService**).
2. Use the **Find Objects** function and/or drag and drop components with an existing display name that you wish to change to an identical value. Note all components listed in the view will be changed in an identical manner.
3. Click **Edit Slot** to bring up the popup **Edit Slot** dialog.
4. In the **Edit Slot** dialog, click the **Property** field control and select **displayNames** from the drop-down list.

At least one of the components listed in the Batch Editor must have the "Readonly" flag cleared on its `displayNames` slot; otherwise `displayNames` is not available in the list of properties.

The **Edit Slot** dialog populates the **New Value** field with the value of the `displayNames` slot for the *first component listed* in the Batch Editor.

Note: This *does not* mean all components listed in the Batch Editor list have identically configured display names. Although if you clicked **OK** now, the Batch Editor would attempt to make this happen.

5. Edit the **New Value** field in the **Edit Slot** dialog with the display name value(s) you wish to set in all components listed in the Batch Editor. Use the proper BNameMap syntax.

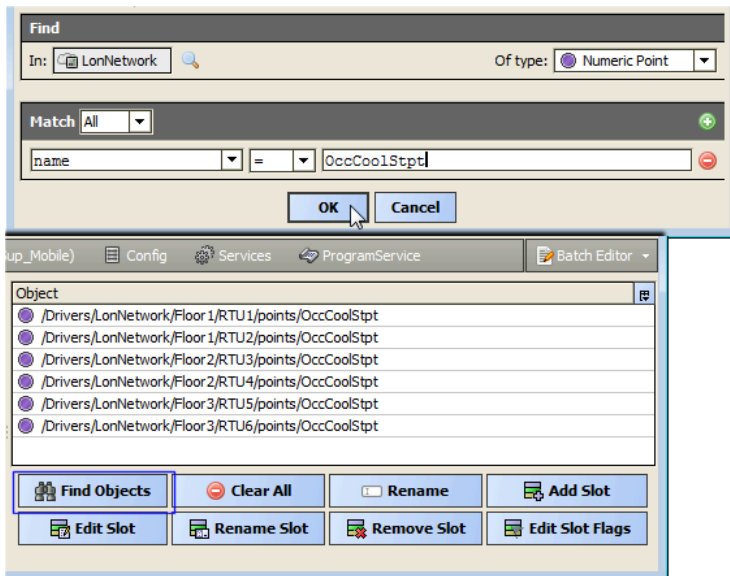
You must enter the semi-colon (;) at the end of each value string, which is required even if entering only one key-value pair. Note that the field editor does not stretch to display all of the text.

6. With the **New Value** field edited to the desired value(s), click **OK**. A **BatchEditor Results** popup replaces the **Edit Slot** dialog. It lists the edit slot results for each component listed in the Batch Editor.
7. Click **OK** to close the **BatchEditor Results** dialog and return to the Batch Editor view.

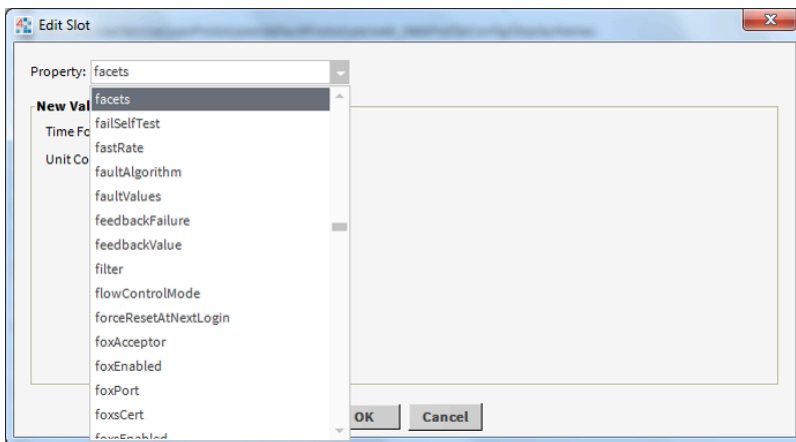
Example batch edit of existing display names

In this example, numeric points named "OccCoolStpt" under the station's LonNetwork are given a display name change.

1. The **Find Objects** function in the **Batch Editor** is used to find the target components.

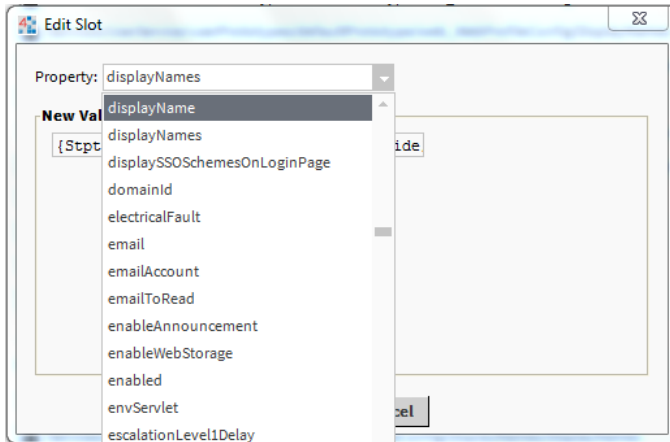


2. The **Edit Slot** button is clicked, and the **Edit Slot** popup dialog appears.

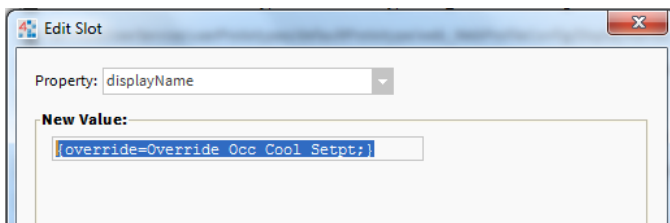


The **displayNames** slot is selected in the **Property** field. Note if all components listed in the **Batch Editor** have the “Readonly” flag set on their `displayNames` slot, the **Property** field is *not available* in the drop-down list, as shown in the preceding figure.

However, if at least one of the components listed has the “Readonly” flag cleared on its `displayNames` slot, this slot is available, as shown in the following figure.

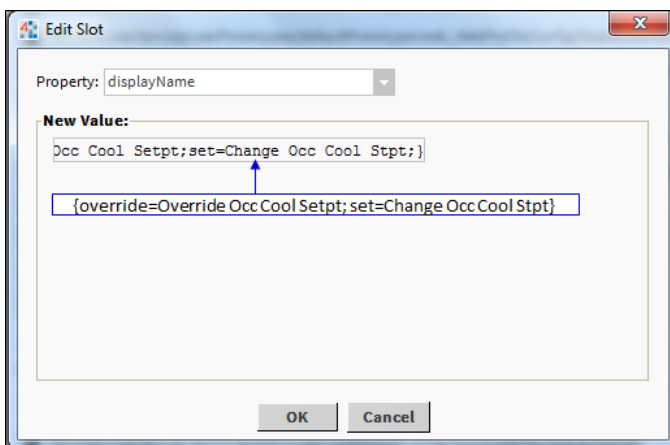


The **New Value** field in the **Edit Slot** dialog initially reflects the current `displayNames` slot value in the first component listed in the Batch Editor.



In this case, the value is `{override=Override Occ Cool Setpt;}`

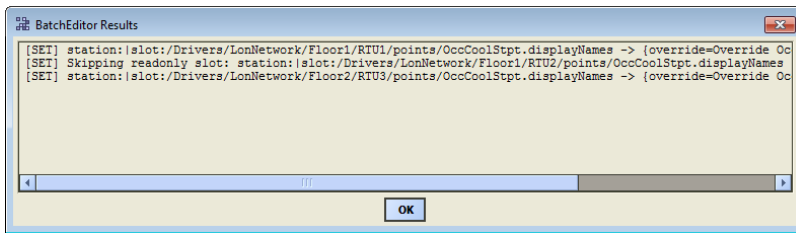
3. The changed value is typed into the **New Value** field.



Note the field editor does not stretch to display all of the text entered. In this case the new value (to be written to all components listed in the Batch Editor) is:

```
{override=Override Occ Cool Setpt;set=Change Occ Cool Stpt;}
```

4. The **OK** button is clicked in the **Edit Slot** dialog, which is replaced by the **BatchEditor Results** popup showing the edit slot results.



In this example with six components originally listed in the Batch Editor, only three components have result lines. This indicates that the other three (missing) components did not have a `displayNames` slot. Of the three components acted upon by the Batch Editor, two actually had the slot edit performed, while one component was *skipped* because its `displayNames` slot had its “Readonly” config flag set.

Parent topic: [Using the Batch Editor for display names](#)

Using the Batch Editor to add display names

The **Batch Editor** view of the station’s **ProgramService** allows you to *add* display names for slots in control points or even child components of container components.

The station must be running and opened in Workbench, with the `program` module installed on the Niagara host. The **ProgramService** should be in the station’s **Services** folder.

CAUTION: Before using the Batch Editor, always *save and backup the station*. It is easy to make errors using the Batch Editor, and there is *no undo*. Therefore in a worst-case scenario, you can always reinstall the saved station from your backup.

Typically, you add the `displayNames` slot to components *without* any display names already assigned. This adds the slot and assigns the display name value(s) in one operation. If a listed component already has a `displayNames` slot, its value is typically overwritten by the new value in this task, unless you had first cleared the “Set if exists” checkbox.

Note: Any `displayNames` slot added by the **Batch Editor** is created *without* any config flags set. This varies from the `displayNames` slot created as a result of the manual **Set the Display Name** command (on component or from slot sheet).

1. In Workbench with the station opened, access the **Batch Editor** (in the Nav tree, expand the **Config, Services** node and double-click **ProgramService**).
2. Use the **Find Objects** function and/or drag and drop components for which you want to add display names to child slots. Note all components listed in the view will be changed in an identical manner.
3. Click **Add Slot** to bring up the popup **Add Slot** dialog.
4. In the **Add Slot** dialog, type `displayNames` in the **New Name** field.
5. In the **New Type** field, ensure `ba_ja` is selected on the left side, and click the right side drop-down control and select **NameMap**.
6. For the checkbox **Set if exists** :
 - Leave checked (the default) if components that already have a `displayNames` slot should have their current value *overwritten* with this new value.
 - Clear (uncheck) if components that already have a `displayNames` slot should *retain their current value*. In this case, the **Batch Editor** does not alter such components.
7. In the **New Value** field, type in the display name key-value pair(s) using the NameMap formatting `{slotName1=displayValue;slotName2=displayValue;} (and so on).`

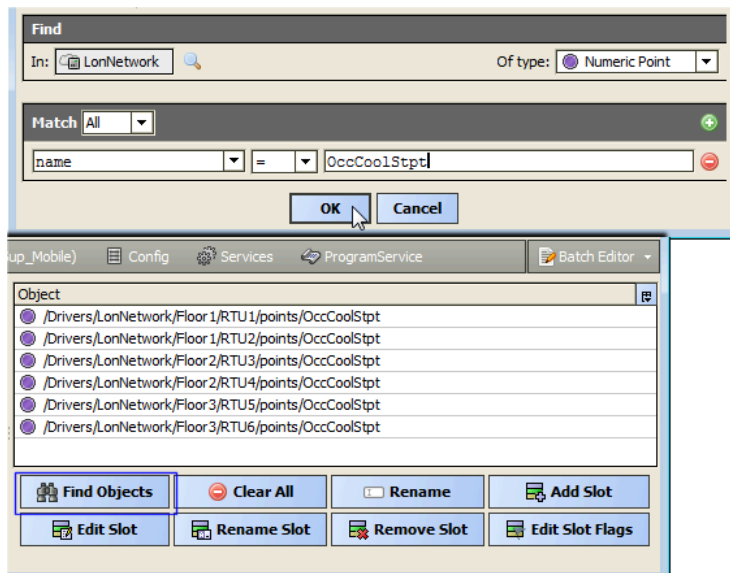
Note: Ensure the value string includes the beginning and ending braces ({ and }), and that each slot key-value pair ends with a semi-colon (;)—even if only a single key-value pair. Also note that the slot name (key) portion is case-sensitive and so much match the actual slot name; for example `emergencyAuto` instead of `Emergency Auto`.

8. When finished in the **Add Slot** dialog, click **OK**.
A **BatchEditor Results** popup dialog replaces the **Add Slot** dialog. It lists the add slot results for each component listed in the Batch Editor (if the “**Set if exists**” option was set); otherwise it lists the results only for those components listed that were previously without a `displayNames` slot.
9. Click **OK** to close the **BatchEditor Results** dialog and return to the Batch Editor view.

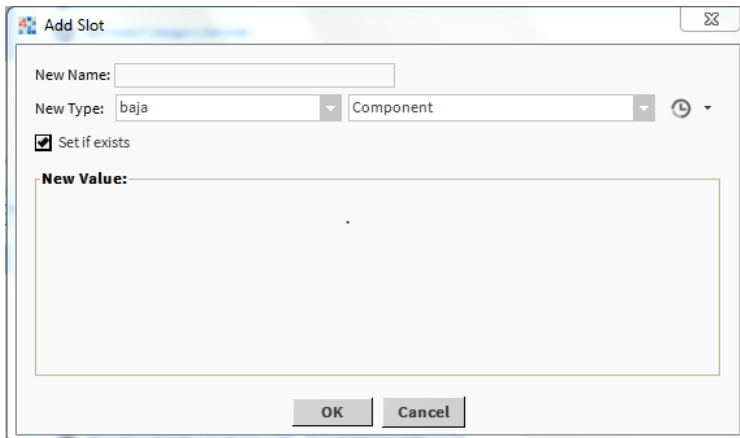
Example batch edit to add display names

In this example, numeric points named “OccCoolStpt” under the station’s **LonNetwork** are given a `displayNames` slot.

1. The **Find Objects** function in the **Batch Editor** is used to find the target components.



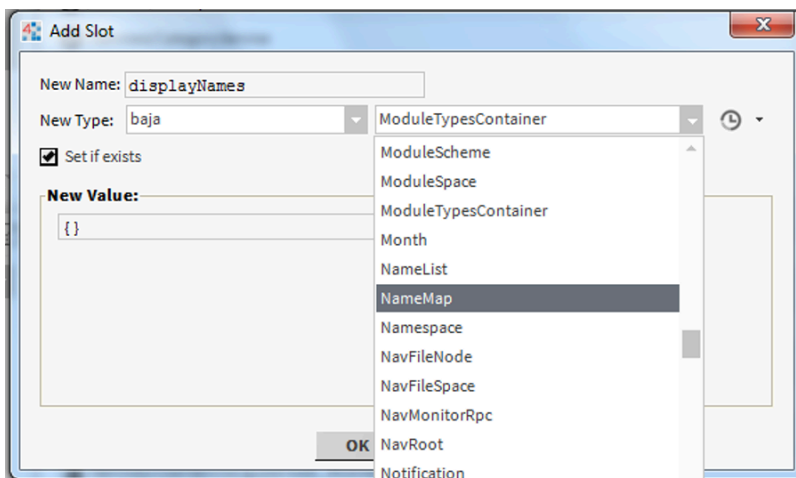
2. The **Add Slot** button is clicked, and the **Add Slot** popup dialog appears.



3. In the **New Name** field, `displayNames` is entered.

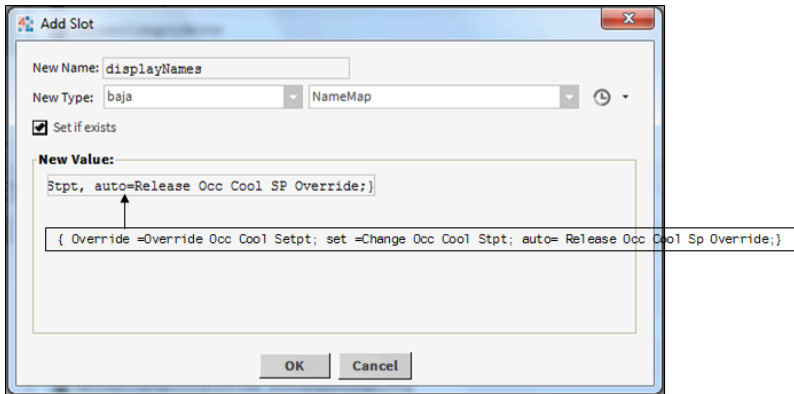
Note: It must be entered exactly like this—plural.

NameMap is selected from the `baja` type drop-down list.



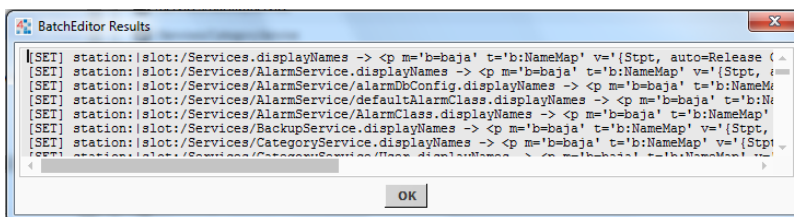
The checkbox **Set if exists** is left set.

4. In the **New Value** field, the `displayNames` name map value is typed in.



In this case, the value is {override=Override Occ Cool Setpt;set=Change Occ Cool Stpt;auto=Release Occ Cool SP Override;}. Note the field editor does not stretch to display all of the text entered.

5. The **OK** button is clicked in the **Add Slot** popup, which is replaced by the **BatchEditor Results** popup showing the edit slot results.



In this example with 6 components originally listed in the **Batch Editor**, 3 components have [SET] result and 3 have [ADD] result lines.

- [SET] indicates the component already had a `displayNames` slot. The new value overwrote the existing value.
- [ADD] indicates the component had no `displayNames` slot. The slot was added with the new value.

Parent topic: [Using the Batch Editor for display names](#)

Using the Batch Editor to change slot flags

The **Batch Editor** lets you batch edit *config flags* for the `displayNames` slot in multiple components. Often this is useful when using the Batch Editor to edit or add display names.

The station must be running and opened in Workbench, with the `program` module installed on the Niagara host. The **ProgramService** must be in the station's **Services** folder.

CAUTION: Before using the Batch Editor, always *save and backup the station*. It is easy to make errors using the Batch Editor, and there is *no undo*. Therefore in a worst-case scenario, you can always reinstall the saved station from your backup.

The Batch Editor allows you to change a *config flag* in a selected slot in multiple components. In the context of display names, this can be useful applied to the `displayNames` slot of any component that has one or more

child slots with display names assigned.

For example:

- To clear (remove) the “Readonly” flag on `displayNames` slots, to allow batch editing.
- To set the “Hidden” flag on batch-added `displayNames` slots, to prevent **displayNames** from appearing on a property sheets.

Optionally, you could also set the “Operator” flag, to allow future operator-level edits of display names, and set the “Readonly” flag (if you are finished batch editing them).

Note: Any `displayNames` slot added by the **Batch Editor** is created *without* any config flags set, unlike the `displayNames` slot created from an initial (manual) “**Set the Display Name**” command.

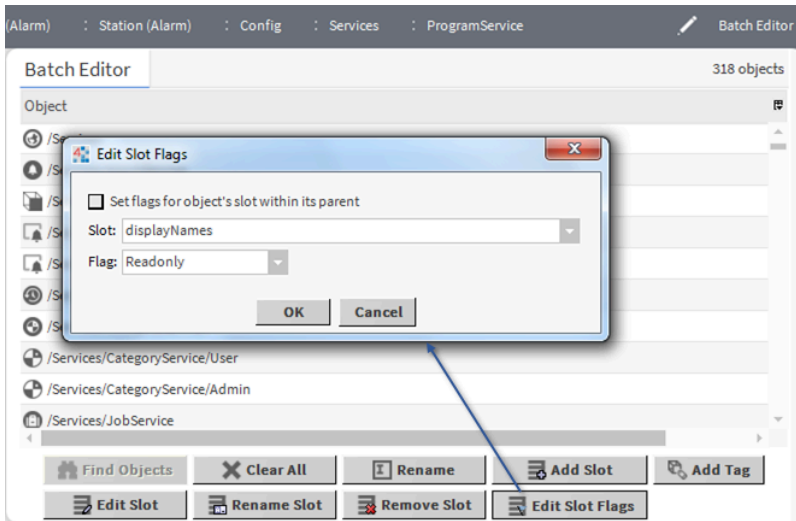
1. In Workbench, look at the slot sheet of components like the ones you wish to change config flags on their `displayNames` slot. Note which letters may appear in the **Flags** column for the `displayNames` slot. Typical letters are `r` (Readonly), `h` (Hidden), `o` (Operator). For example `rho` means all of these config flags are set, whereas only `h` means just the Hidden flag is set. No letters means all config flags are cleared (removed).
2. Access the **Batch Editor** (in the Nav tree, expand the Config, Services node and double-click **ProgramService**).
3. Use the **Find Objects** function and/or drag and drop components with an existing `displayNames` slot that you wish to change a config flag. Note all components listed in the view will be changed in an identical manner.
4. Click **Edit Slot Flags** to bring up the popup **Edit Slot Flags** dialog.
5. In the **Edit Slot Flags** dialog:
 - a. Leave the checkbox *cleared* for **Set flags for object’s slot within its parent**.
 - b. In the **Slot** field select **displayNames**
 - c. In the **Flag** field select the appropriate flag. For example, `Readonly` (to remove) or `Hidden` (to set).
 - d. Click the **Set Flag** or **Remove Flag** control to select.
 - e. Click **OK** to issue the command.

A **BatchEditor Results** popup replaces the **Edit Slot Flags** popup. It lists the results of the config flag batch edit.
6. Click **OK** to close the results popup and return to the Batch Editor.

Example batch edit clearing of “Readonly” config flag on points

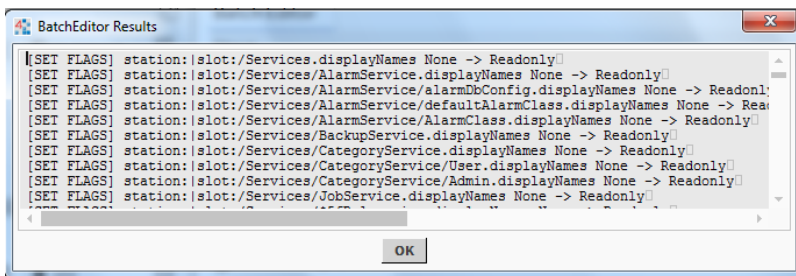
In this example, several points originally had action display names manually changed from their slot sheet. As a result, their `displayNames` slot has config flags set for “Readonly”, “Hidden”, and “Operator” (`rho`). In order to use the Batch Editor to further edit display names for the slots of these points, the “Readonly” config flag must be cleared (removed).

1. The points are dragged into the **Batch Editor** and the **Edit Slot Flags** button is clicked.



In the **Edit Slot Flags** popup, `displayName` is selected in the **Slot** field, and in **Flag** field the `Readonly` entry is selected. The **Remove Flag** control is selected.

2. The **OK** button is clicked, and the **Edit Slot Flags** popup is replaced by the **BatchEditor Results** popup, showing the edit slot flag results.



In this example, the **BatchEditor Results** popup shows the batch edit removed the `Readonly` flag for all eight of the points in the Batch Editor.

Parent topic: [Using the Batch Editor for display names](#)

Troubleshooting batch edited display names

Using the **Batch Editor** for adjusting display names can save time; however, it is easy to overlook some things that prevent success. This section lists some common pitfalls and recommendations when using this method.

Key concepts

Keep in mind the following when using the Batch Editor:

- The Batch Editor runs against *all listed objects* in the view when you select an action, whether or not any appear selected (highlighted). Therefore, be sure to select and *remove* any unwanted objects in the view *before* running any action.
- Display names are stored on the *parent* of the target slot or component.
- The “BNameMap” value of the parent’s `displayName` slot (if present) contains all explicitly assigned

display names for child slots or components. Editing from the Batch Editor *replaces* this entire value with your new value—it does not “append” to any existing value.

- If batch editing display names that were originally manually set, the corresponding `displayNames` slot is likely to have its “Readonly” config flag set. This prevents any batch edit operation on it. In this case, use the Batch Editor to edit slot flags on the `displayNames` slot (clear the “Readonly” flag).

Common pitfalls

The following are some common pitfalls when using the Batch Editor against display names:

- Added `displayNames` values appear to be added without errors; however the target slots do not appear to have the display name values.

This can happen when “automatic display name” values are entered in `BNameMap` “key value” pairs, instead of actual slot names. For example, entering: `{Set=Change Setpoint;}` instead of `{set=Change Setpoint;}`. In this case a batch edit would run without errors, and a `displayNames` slot would be created. However, none of the `set` slots would have their display name value bolded (changed). See [BNameMap syntax](#).

- Omitting the ending semi-colon after each `BNameMap` “key-value” pair of a `displayNames` slot. This is required even if only one pair (display name) is entered.
- Entering the slot to be added in the Batch Editor as `displayName` instead of `displayNames`. The Batch Editor looks specifically for a `displayNames` slot, and if misspelled without the trailing “s”, display names are not affected.

Parent topic: [Using the Batch Editor for display names](#)

Niagara R2 to AX via oBIX

This document provides information on integrating Niagara R2 stations into the AX system using oBIX technology, and assumes that you are knowledgeable about the Niagara engineering used in both system types (often abbreviated simply “R2” and “AX”). Other reference details about Niagara oBIX implementations can be found in the *oBIX Guide – AX* and the *Niagara Release 2 oBIX User Guide*, as well as the comprehensive public specification documents found at OASIS (at the time of this document) at the following URL:

<http://www.oasis-open.org/committees/download.php/21462/obix-1.0-cs-01.zip>.

Note: It is recommended that you first read the other Niagara / AX oBIX documents to become familiar with oBIX terms like “lobby,” as well as the general operation of the two oBIX drivers.

- **R2 to AX oBIX Overview**

In an AX station you can use the obixDriver for client-side oBIX access of remote R2 stations to mix R2 and AX data together. In the most-anticipated scenario, the AX Supervisor will be added to a job that already has R2 controllers and an R2 Supervisor, along with additional AX controllers. A possible eventual goal of the AX Supervisor is to replace the R2 Web Supervisor. This document focuses on the AX Supervisor, and notes areas where R2 to AX “switchover” has limitations, or may require additional engineering.

- **R2 station engineering**

The following sections apply to R2 station and host engineering:

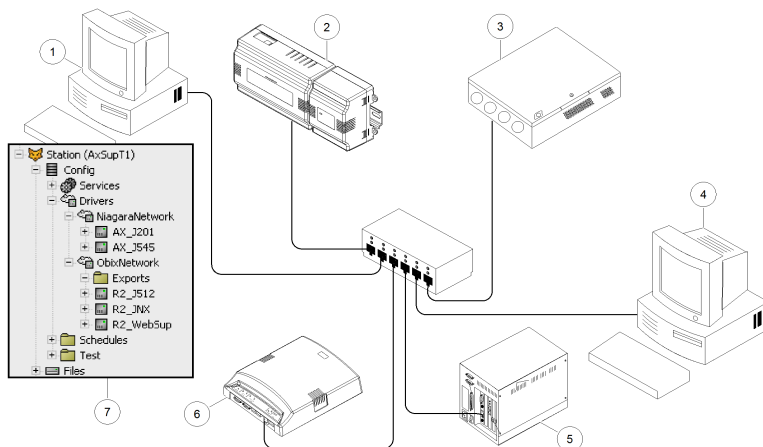
- **AxSupervisor engineering**

The following sections describe Niagara Supervisor engineering topics that relate to integrating R2 stations:

R2 to AX oBIX Overview

In an AX station you can use the obixDriver for client-side oBIX access of remote R2 stations to mix R2 and AX data together. In the most-anticipated scenario, the AX Supervisor will be added to a job that already has R2 controllers and an R2 Supervisor, along with additional AX controllers. A possible eventual goal of the AX Supervisor is to replace the R2 Web Supervisor. This document focuses on the AX Supervisor, and notes areas where R2 to AX “switchover” has limitations, or may require additional engineering.

Figure 1. AX Supervisor access to R2 stations and AX controllers using different network types



| | | |
|---|----|--------------------|
| 1 | AX | Supervisor |
| 2 | AX | JACE-2 |
| 3 | AX | JACE-545 |
| 4 | R2 | Supervisor |
| 5 | R2 | JACE |
| 6 | R2 | JACE |
| 7 | AX | Supervisor Station |

- [Requirements](#)
- [Engineering summary](#)
Most engineering is expected to occur in the AX Supervisor station where:

Parent topic: [Niagara R2 to AX via oBIX](#)

Requirements

- Each R2 host (JACE, Web Supervisor) should be running Niagara 2.301.522 or later, have the obix module 2.301.527c.beta or later installed, and has its license enabled for the “obix” feature. Your JDE (WorkPlace Pro) should also be at 2.301.522 or later.

Optionally, each R2 host can also have an “obixInternal.properties” file in its nre\lib folder. This helps when doing Obix proxy point discovers from the NiagaraAX client side, in coordination with the NiagaraAX feature to hide rarely accessed (internal) properties. See “About Discover “Include” options” on page 2-10 for related details.

- The NiagaraAX Supervisor host must be running NiagaraAX3.1.31 or later, have the installed modules:
 - obix 3.3.22 or later
 - obixDriver 3.3.28.2 or later

The NiagaraAX Supervisor also requires the “obixDriver” feature in its license. Any NiagaraAX controller that needs direct Obix client access to any R2 controller also has the same requirements.

- [Loading effects of oBIX integration](#)
Please be aware that oBIX creates an additional load on R2 stations. Tests indicate the following:

Parent topic: [R2 to AX oBIX Overview](#)

Loading effects of oBIX integration

Please be aware that oBIX creates an additional load on R2 stations. Tests indicate the following:

- A VxWorks (controller) R2 station with 40 percent idle time and 500,000 resources consumed can accommodate 1000 AX Obix proxies in a watch.
- A VxWorks (controller) R2 station with 30 percent idle time and 500,000 resources consumed can accommodate 500 AX Obix proxies in a watch.

Inspect idle time by pointing a browser to: <http://ipAddress:3011/system/spy>

Determine resource count by pointing a browser to: <http://ipAddress/prism/resources>

Please note that the above results depend on how the AX client is reading the values, that is, is it polling all of the points all of the time, or only some, etc. “Your results may vary.” In addition, the watch interval (adjusted on the AX client side) plays a role in the loading of an R2 station.

For related details, see “Pre-integration R2 station changes” on page 2-3, and “Client polling timing” on page 2-15.

Parent topic: [Requirements](#)

Engineering summary

Most engineering is expected to occur in the AX Supervisor station where:

- All data from remote R2 stations is modeled under the Drivers/ObixNetwork, using the basic familiar AX driver architecture (e.g. network/devices/Points/proxyPoints). See “AX station engineering summary”.
- All data from remote AX controller is modeled under the Drivers/NiagaraNetwork, as for any AX Supervisor. Refer to “About the Niagara Network” in the *NiagaraAX Drivers Guide* for more details.
- [R2 station engineering summary](#)
Make pre-integration changes to the R2 station(s), if necessary. Then from the tridiumx/obix jar, add the ObixService to the Services container of any R2 station to be integrated.
- [AX station engineering summary](#)
Add an ObixNetwork in the AX Supervisor’s Drivers container, either using the New command in the Driver Manager view, or by opening the obixDriver palette and copying/dragging the network. Then, under this network you manually add one R2ObixClient device component for each R2 host (station) to be included.

Parent topic: [R2 to AX oBIX Overview](#)

R2 station engineering summary

Make pre-integration changes to the R2 station(s), if necessary. Then from the tridiumx/obix jar, add the ObixService to the Services container of any R2 station to be integrated.

Use of the other R2 “export” objects in the tridiumx/obix jar is optional, and may not apply. Depending on other integration features into AX, other station configuration changes may be necessary regarding slaved Schedule objects, and the station’s LogService and/or NotificationService.

For more details, see “R2 station engineering”.

Note: See the *Niagara Release 2 oBIX User Guide* for details on copying the obixInternal.properties file to remote R2 controllers, as well as other R2-specific topics.

Parent topic: [Engineering summary](#)

AX station engineering summary

Add an ObixNetwork in the AX Supervisor’s Drivers container, either using the New command in the Driver Manager view, or by opening the obixDriver palette and copying/dragging the network. Then, under this network you manually add one R2ObixClient device component for each R2 host (station) to be included.

Once you configure an R2ObixClient with a few settings, it can connect (attach) to the host R2 oBIX server. Then you can “learn” an object space tree for that station, which has an expandable root node named the “lobby” (an oBIX term). The root lobby node appears in the Discovered pane of the manager view for each of the device’s extensions, notably Points, Alarms, Histories, and Schedules (all default device extensions).

For more details, see “AX Supervisor engineering” on page 2-7.

Parent topic: [Engineering summary](#)

R2 station engineering

The following sections apply to R2 station and host engineering:

- Pre-integration R2 station changes
- Add ObixService
- R2 ObixExport objects
- Other possible R2 host changes
- [Pre-integration R2 station changes](#)
Before beginning an oBIX integration, consider R2 station changes that can improve performance. The following changes apply:
- [Add ObixService](#)
For each R2 station to be integrated, you open it in the JDE, open the Local Library and expand the `tridiumx/obix` jar, and copy and paste the **ObixService** into the station’s Services container. No further configuration of that service is needed; however, you must restart that station for it to become an “oBIX server.” R2 object data from the station is then immediately available from the AX station running the Obix driver.
- [Other possible R2 station configuration changes](#)
Apart from adding the ObixService and possible use of Obix “export” objects (all copied from the `tridiumx/obix` jar), the following additional configuration changes may be necessary in R2 stations, depending on the intended transition of an R2 Web Supervisor to AX Supervisor.
- [Other possible R2 host changes](#)
After the oBIX integration, when connecting to an R2 host with the AX client, if you receive errors similar to:

Parent topic: [Niagara R2 to AX via oBIX](#)

Pre-integration R2 station changes

Before beginning an oBIX integration, consider R2 station changes that can improve performance. The following changes apply:

- Reducing R2 station resource count
- Reducing R2 swid lengths
- [Reducing R2 station resource count](#)
Typically, the reason for integrating an R2 controller in an AX Supervisor is to provide access to it via AX PxPages on the AX Supervisor. With this goal in mind, in many cases the R2 station’s resource count can be reduced by removing its GxPages and associated Gx objects. For a heavily loaded R2 station, reducing its station size may be the only way of accommodating the R2 oBIX server.
- [Reducing R2 swid lengths](#)
The depth of the R2 tree is of importance in relation to the performance of the AX Client. Obviously,

more data being transmitted from the R2 Server to the AX Client increases the time between updates. Application developers typically would like to keep re-engineering of the R2 station database to a minimum. However, simple name changes to shorten R2 swids (system wide identifiers) can result in a large benefit.

- [Example](#)
Original swid:

Parent topic: [R2 station engineering](#)

Reducing R2 station resource count

Typically, the reason for integrating an R2 controller in an AX Supervisor is to provide access to it via AX PxPages on the AX Supervisor. With this goal in mind, in many cases the R2 station's resource count can be reduced by removing its GxPages and associated Gx objects. For a heavily loaded R2 station, reducing its station size may be the only way of accommodating the R2 oBIX server.

Parent topic: [Pre-integration R2 station changes](#)

Reducing R2 swid lengths

The depth of the R2 tree is of importance in relation to the performance of the AX Client. Obviously, more data being transmitted from the R2 Server to the AX Client increases the time between updates. Application developers typically would like to keep re-engineering of the R2 station database to a minimum. However, simple name changes to shorten R2 swids (system wide identifiers) can result in a large benefit.

Parent topic: [Pre-integration R2 station changes](#)

Example

Original swid:

```
/AcmeBuildingOneFirstFloor/LonTrunk/AirHandlingUnit1/VariableAirVolumeUnits/  
Boxes/Room101/Room101VA
```

Shortened swid:

```
/Acme1stFlr/LT/AHU1/VAVs/Boxes/Rm101/Rm101
```

In addition to increasing the performance of the data transfer, reducing the length of a swid reduces the amount of memory required to serve up the data. This topic reflects a tendency to sometimes "over-engineer" a station database by creating unnecessarily deep hierarchies. Of course some hierarchy is necessary, but more is not necessarily better...

Parent topic: [Pre-integration R2 station changes](#)

Add ObixService

For each R2 station to be integrated, you open it in the JDE, open the Local Library and expand the `tridiumx/obix` jar, and copy and paste the **ObixService** into the station's Services container. No further configuration of that service is needed; however, you must restart that station for it to become an "oBIX server." R2 object data from the station is then immediately available from the AX station running the Obix driver.

Note you can quickly verify if an R2 station is operating as an oBIX server. Simply open a web browser connection to that station, using the syntax

`http://<host>[:port]/obix`

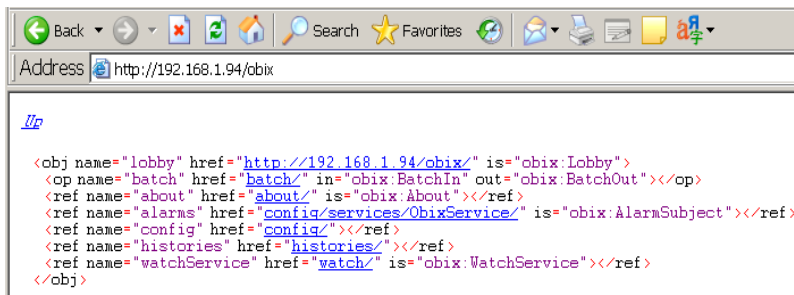
where <host> is IP address or hostname, and [:port] is optional (if omitted, assumed as 80).

For example: `http://192.168.1.94/obix` for a typically-configured station at that IP address,

or `http://192.168.1.75:85/obix` for a station running on httpPort 85 on a host at that IP address.

As shown, after you login with station credentials you see an HTML representation of the station's oBIX lobby, including hyperlinks to traverse into the object tree structure.

Figure 1. Example browser connection to confirm R2 station oBIX server operation



Note R2 oBIX is server only. Client access of remote oBIX servers is unavailable—no R2 “shadow objects” exist to get remote oBIX data (e.g. from the AX station). This simplifies engineering on the R2 side.

Note: For this reason, the AX Supervisor station’s **“Exports”** folder under its ObixNetwork is not used when integrating Niagara R2 stations as ObixClients. However, AX to R2 schedule exports are possible using a different method.

- [R2 ObixExport objects](#)

Parent topic: [R2 station engineering](#)

R2 ObixExport objects

Note: Use of these objects is entirely optional. Many R2 to AX oBIX integrations have not used them, as standard AX Obix proxy points for these specific R2 object types provide “right click command access”, along with value and status. Use is necessary only to permit an “interstation control link”, from AX to the R2 station.

The three “export” objects in the tridiumx/obix jar are available if you want to allow “link control” writes from AX to R2 objects, for example to the “priorityArray” input of an AnalogOutput, BinaryOutput, or MultistateOutput object. In this case, you copy one of objects from the R2 tridiumx/obix jar and paste it into the station, linking its output into the priorityArray input of the R2 object being controlled.

If controlling an AnalogOutput, BinaryOutput, or MultistateOutput, another link is also required—from the statusOutput of the controlled object back to the “feedbackValue” input (fln) of the export object.

Figure 1. Example ObixAnalogExport object copied into station for linking into AnalogOutput object

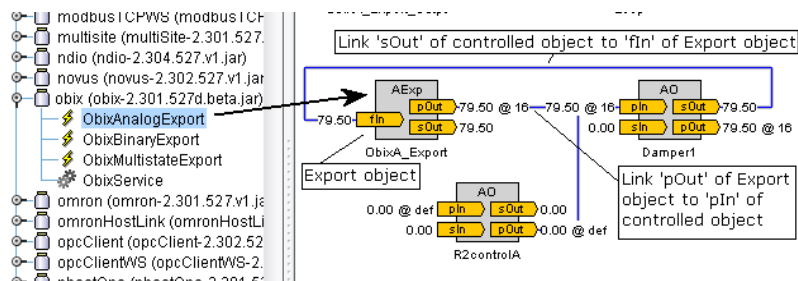


Figure 2-3 shows an example of both links, where an ObixAnalogExport object controls an AnalogOutput (“Damper1”).

Figure 2. ObixBinaryExport object copied into station for linking into BinaryOutput (BO) object

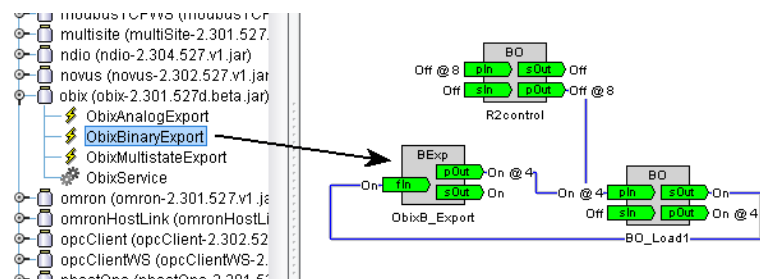


Figure 2-4 shows a similarly-linked ObixBinaryExport object used for linking into a BinaryOutput object, “BO_Load1”.

You can also use an export object to link to a “status” type input of an R2 object, for example an input on a Math object (“FloatStatusType”, using an ObixAnalogExport) or on a Logic object (“BooleanStatusType”, using an ObixBinaryExport object). In this case, you simply link the statusOutput (sOut) of the export object into the statusInput of the R2 object, and need not link the “feedbackValue” input of the export object.

Figure 3. ObixAnalogExport object copied into station for use as “statusInput” type value

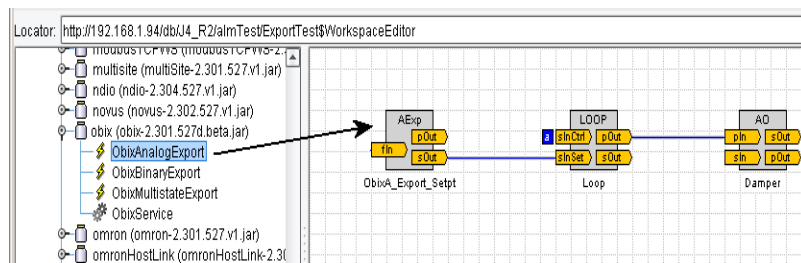


Figure 2-5 shows an example where an ObixAnalogExport object is used for setpoint control of an R2 Loop object, linked into the Loop’s “sInSet” input.

For any R2 export object you use, you must set up several Config properties, described in the next section, “R2 Obix Export object properties”. Then in the AX station, you can “discover” this object within station’s ObixClient “lobby,” and add a writable Obix proxy point for it. This allows you to link other AX station logic into the proxy point, as well as invoke actions on that same point.

R2 Obix Export object properties

All three of the R2 Obix export object have these common properties, as found on tabs in their property sheet:

Status

On the Status tab of an Obix export object, find these read-only properties

- lastWrite — The last value written by the oBIX Client.
- lastWriteTimestamp — The time of the last client write.

Config

On the Config tab of an Obix export object, specify these values accordingly:

- priority — The priority to be used for writing values at the prioritizedOutput (defaults to 16).
- units (if ObixAnalogExport), or

activeInactiveText (if ObixBinaryExport), or

stateText (if ObixMultistateExport) — In any type of export object, set units, etc. to mirror the units for the linked input on the controlled R2 object.

Note: An ObixAnalogExport object has these additional Config properties, described below (see Figure below). Use of these “limit” type properties is optional.

- highLimit — The maximum value that can be written by the oBIX client.
- lowLimit — The minimum value that can be written by the oBIX client.
- limitEnabled — If set to true, high and low limits are enforced on client writes (default is false).

Figure 4. Config properties for ObixAnalogExport, including “limit” type properties

The screenshot shows the 'Config' tab of a configuration window titled 'portTest/ObixA_Export_Setpt\$Properties'. The window has five tabs: Status, Config (selected), Visual, Engineering, and Security. The Config tab contains the following properties and values:

| | |
|---------------------|-------------------------------|
| executionParameters | freq: normal (dropdown) |
| | order: processor (dropdown) |
| foreignAddress | -1 (text field) |
| membershipGroups | niagaraR2 (text field) |
| highLimit (°F) | 80.0 (text field) |
| lowLimit (°F) | 60.0 (text field) |
| limitEnabled | True (dropdown) |
| priority | Schedule (16) (dropdown) |
| units | Temperature (dropdown) |
| | degrees_Fahrenheit (dropdown) |

Any “outside of limit” value that a client attempts write to the R2 object input is rejected, and the last “in-limit” value is retained. For related details on the AX side, see “AnalogExport Limit Notes” on page 2-14.

Engineering

On the Engineering tab of an ObixExport object, find these read-only properties

- `feedbackValue` — If linked, the value returned for client read requests on this object.
- `prioritizedOutput` — Value being written on this “pOut” output.
- `statusOutput` — Value being written on this “sOut” output.

Parent topic: [Add ObixService](#)

Other possible R2 station configuration changes

Apart from adding the ObixService and possible use of Obix “export” objects (all copied from the `tridiumx/obix.jar`), the following additional configuration changes may be necessary in R2 stations, depending on the intended transition of an R2 Web Supervisor to AX Supervisor.

Note: Before making any of the changes below, please read the related AX sections in this document to understand current limitations. Currently, there is no known utility to “convert” the existing application database (`appdb`) of an R2 Web Supervisor into a format that can be “merged” into the existing AXSupervisors collection of histories and alarm database.

- If mastering R2 Schedule objects from the AX Supervisor (or another AX controller), and they are currently “slaved” to another R2 Schedule, you need to first unlink those slaved R2 Schedules (remove `externalSubscription` at “`slaveIn`” input). For more information on engineering for the “AX side,” see “AX Supervisor engineering” on page 2-7.
- Depending on how log archiving is to continue for any R2 JACE station, you may wish to change the setup of the station’s `LogService`, with the following Config tab properties:
 - `archiveMode` — from “`archive_remote`” to “`archive_local`”.
 - `archiveAddress` — uncheck Supervisor entry
- Depending on alarm/alert management is to continue for any R2 station, you may wish to change the setup of the station’s `NotificationService`, with the following Config tab properties:
 - `alarmArchiveAddress` — uncheck Supervisor entry
 - `archiveMode` — from “`archiveRemote`” to “`archive_local`” (if the JACE-403/JACE-545, “`archive_local_no_SQL`”).

Parent topic: [R2 station engineering](#)

Other possible R2 host changes

After the oBIX integration, when connecting to an R2 host with the AX client, if you receive errors similar to:

```
HTTP Error 503:Service Unavailable
```

this indicates that all available web service threads on the R2 host are currently being used.

Typically, you can fix this by editing an entry in the `system.properties` file on the R2 host:

```
webServer.threadPoolSize=n
```

The value of this thread pool defaults to 15 in an R2 controller, but can safely be increased to 30.

Parent topic: [R2 station engineering](#)

AxSupervisor engineering

The following sections describe Niagara Supervisor engineering topics that relate to integrating R2 stations:

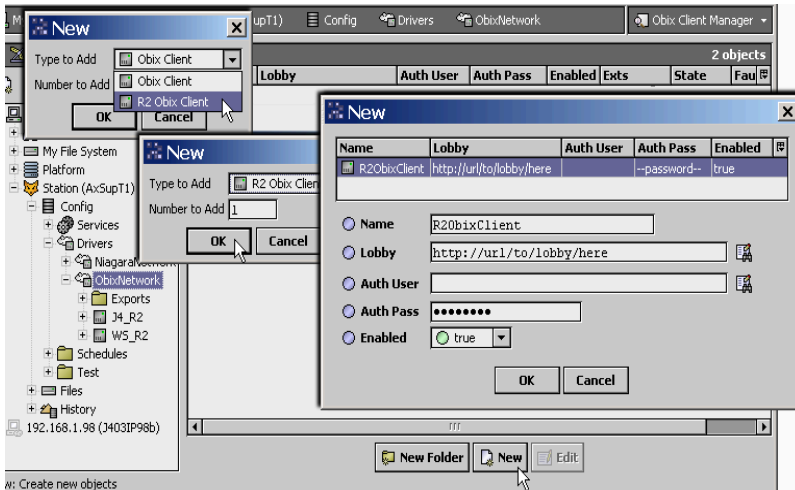
- ObixNetwork and R2ObixClient devices
- R2ObixClient Points
- R2ObixClient Histories (logs and archives)
- R2ObixClient Alarms
- R2ObixClient R2 Schedule exports
- [ObixNetwork and R2ObixClient devices](#)
In the AX Supervisor you add a single ObixNetwork under Drivers, then manually add R2ObixClient devices, where each represents an R2 station. For each new R2ObixClient, you enter a few properties in the **New** dialog, shown with non-working default values below.
- [R2ObixClient Points](#)
The Points extension of the R2ObixClient device is where most AX engineering is anticipated—double-click the **Points** icon of an R2ObixClient to see the Obix Point Manager. Then click **Discover**.
- [About Obix proxy points](#)
Added Obix proxy points often resemble Niagara proxy points in the following ways:
- [R2ObixClient Histories \(logs and archives\)](#)
The Histories extension of an R2ObixClient is where you can import data from the R2 station's log objects, as well as existing archives from its appdb (if it has the DatabaseService), by adding history import descriptors. The default view of Histories is the ObixHistoryManager (Figure 2-20), where you specify which discovered logs and archives you want to import, using the familiar oBIX “lobby” tree in the Discovered pane.
- [R2ObixClient Alarms](#)
The Alarms extension of an R2ObixClient for the R2 station allows you to add “alarm feed” sources, such that native R2 events (alarms and alerts) in the station can be visible in the AX station's AlarmService.
- [R2ObixClient R2 Schedule exports](#)
By default, each R2ObixClient has 4 device extensions: Alarms, Histories, Points and Schedules. Expand the R2ObixClient and double-click **Schedules** (R2ScheduleDeviceExt) for the Obix Schedule Manager view, and then perform a **Discover**. Expand the **config** branch of the **lobby** to locate target Schedules in the R2 station. Schedules are added as ObixScheduleExport descriptors, as shown in Figure 2-27.

Parent topic: [Niagara R2 to AX via oBIX](#)

ObixNetwork and R2ObixClient devices

In the AX Supervisor you add a single ObixNetwork under Drivers, then manually add R2ObixClient devices, where each represents an R2 station. For each new R2ObixClient, you enter a few properties in the **New** dialog, shown with non-working default values below.

Figure 1. Add R2ObixClients using New button in Obix Client Manager view of ObixNetwork



Note: As shown above, there are two types of “client device” choices: ObixClient and R2ObixClient. Always select R2ObixClient for any R2 station, as it provides additional capabilities.

Also, note the general “client/server” naming in AX follows a “convention” used in some other drivers, for example the OPC driver and Modbus drivers, where a “client” device actually represents a server device (here, an oBIX server), and associated AX components are named “client” because a client connection is used to retrieve data.

Set properties in the New dialog as follows:

- Name

The AX name for the device component—by convention, enter the R2 station name. It must be unique among other child devices, and if using history ID defaults, also recommended to be unique from any NiagaraStation names (under the station’s NiagaraNetwork, for remote AX stations).

- Lobby

The URI to the root of the R2 station’s oBIX server object tree (lobby), using syntax:

`http://<host>[:port]/obix`

where <host> is IP address or hostname, and [:port] is optional (if omitted, assumed as 80).

For example: `http://192.168.1.94/obix` for a typically-configured station at that IP address, or `http://192.168.1.75:85/obix` for a station running on httpPort 85 on a host at that IP address.

- Auth User

Enter user name in that R2 station, typically with all admin-level privileges for most security groups.

Note: On the AX client side, if you attempt object writes without this user having the necessary security rights for those objects (for example a command, or modify a property), it results in an "HTTP Error 401:Access Denied" error.

- Auth Pass

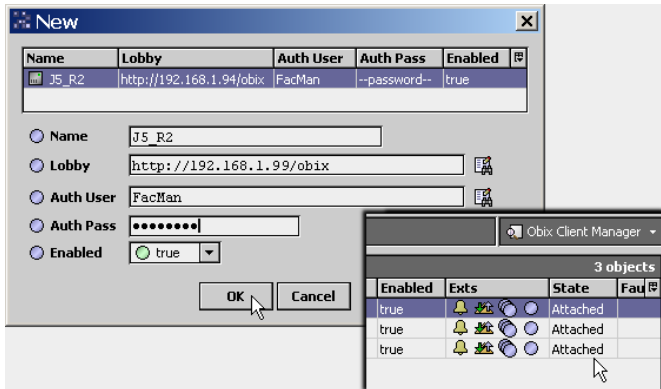
Enter password for that user account in the R2 station.

- Enabled

Defaults to true (must be true to attempt communications and operate).

After entering data and clicking OK, the device is added to the database, and the “State” column value for the new row quickly changes from “Detached” to “Attaching”, and finally to “Attached”, as shown above.

Figure 2. Entering new ObixClient and subsequent State change to Attached



Note: (Troubleshooting) If after entering the R2ObixClient it remains Detached, review its Lobby syntax, including IP address of the R2 controller (open a command prompt window and issue a `Ping` command to that IP address). Also, verify the R2 station user credentials for Auth User and Auth Pass are correct. Note also that you should be able to open a browser connection to the R2 station using the URI entered for Lobby, and after entering user credentials, see its oBIX lobby.

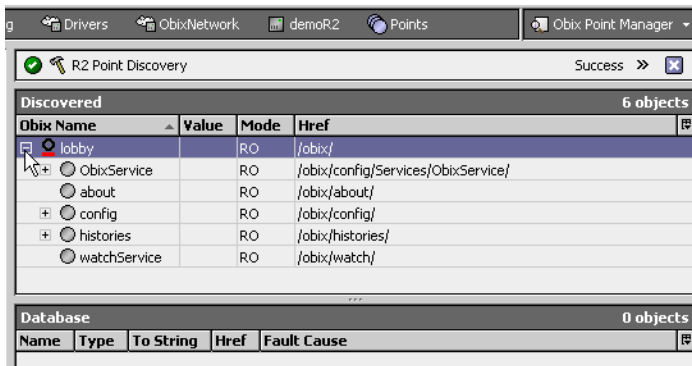
Parent topic: [AxSupervisor engineering](#)

R2ObixClient Points

The Points extension of the R2ObixClient device is where most AX engineering is anticipated—double-click the **Points** icon of an R2ObixClient to see the Obix Point Manager. Then click **Discover**.

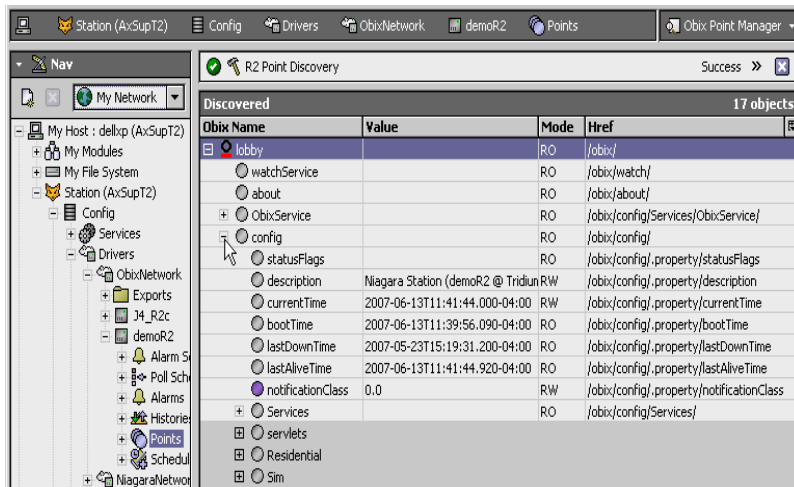
In the discovered pane of the Obix Point Manager, expand the root “lobby” to see the tree organization, as shown below. Items of practical interest for proxy points are under the “**config**” branch.

Figure 1. Top-level lobby structure in R2 station



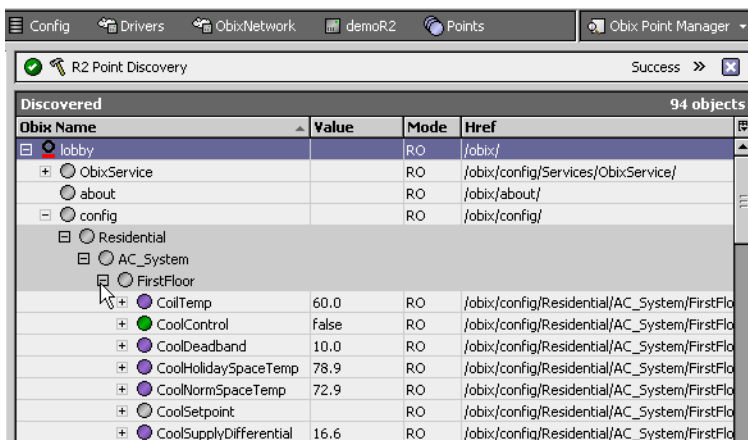
Expand the “config” branch of the lobby to find proxy point candidates. The config hierarchy reflects the R2 station’s object hierarchy, including a row for each object property, and expandable rows for child objects. The highest-level config node is the Station object. So when you first expand config, at the top are the properties found on the various tabs of the property sheet for the Station (when using the R2 JDE).

Figure 2. ObixPointManager with lobby expanded to show discovered proxy point candidates



Note: As needed, click on column headers in the Discovered pane to sort as ascending ▲ or descending ▼, in ASCII character order. For example, if you have the Obix Name column sorted ascending, when you expand items containers will be at top, and properties at bottom, as shown.

Figure 2. Use column header sorts and expand containers as needed to find R2 objects and properties



R2 container objects appear in the discovered lobby as expandable gray rows (Figure 2-11), which you can add as Obix Point Folders by double-clicking. See “About Discover “Include” options” on page 2-10.

Typical expandable target containers contain R2 “shadow objects” or related objects in the R2 station—often the same objects that are linked to R2 Gx objects in GxPages for real-time values and commands.

Note: Although each discovered object is available as a “root” node, you should always expand discovered objects to specify a property for any proxy point. For related details, see “Specify the property” on page 2-11.

- [About Discover “Include” options](#)

Starting in the AX-3.2, the Obix R2 points discovery behavior was improved, and new properties were added to the **Points** device extension (R2PointsDeviceExt). Improvements made include the following:

- **Specify the property**
When creating proxy points under an R2ObixClient, always select the specific property you want to display, rather than the parent root object (node) itself. For example, expand the entry for a BinaryOutput object and select its statusOutput property (Figure 2-13).
- **Add notes for R2 Obix proxy points**
When adding R2 Obix proxy points from the discovered lobby, and selecting a property (recommended), note that currently the default **Name** is the same as the selected R2 property name, such as “statusOutput”, “prioritizedOutput”, and so on. In addition, Facets show default values. See figure below.

Parent topic: [AxSupervisor engineering](#)

About Discover “Include” options

Starting in the AX-3.2, the Obix R2 points discovery behavior was improved, and new properties were added to the **Points** device extension (R2PointsDeviceExt). Improvements made include the following:

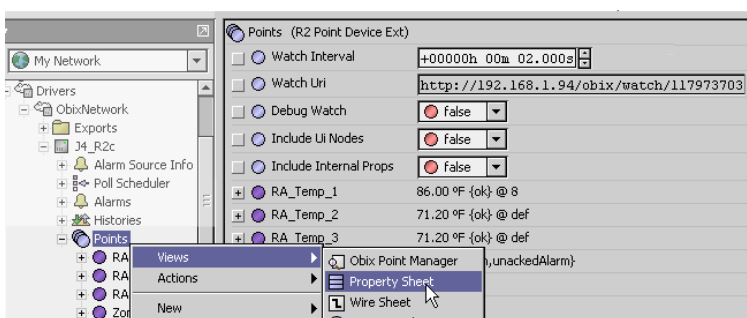
- All R2 container-type objects, including types Container, Bundle, PollOnDemand, and PollAlways appear in the discovered lobby tree as “groups” (gray rows)—you still expand them to see child objects. If you double-click (to add), an Obix Point Folder is immediately created, without an intervening popup dialog.
- By default, all R2 Gx objects (GxText, GxFan, and so on) are omitted from the discovered lobby tree, although GxPage containers still appear. This is controlled by the setting of the property of the Points extension of the parent R2ObixClient: **Include Ui Nodes** (default value is `false`). It is recommended to be left at default.
- By default (if the associated R2 host has the file `nre/lib/obixInternal.properties` installed), all named properties of R2 objects are globally omitted from the AX discovered lobby tree, typically those rarely adjusted (if ever) during normal R2 configuration. On the AX side, this is controlled by the setting of the property of the Points extension of the parent R2ObixClient: **Include Internal Props** (default value is `false`). If you need AX access to these properties, set this slot to `true`.

Note the R2 “shipped” version of `obixInternal.properties` contains a minimal list of properties—you can edit this file to add more properties, one per line. Driver-specific properties, such as LON props and so on, are not included.

If the R2 host does not have the `obixInternal.properties` file installed, or its station has not been restarted since that file was installed, the value of the “Include Internal Props” property makes no difference—all properties are included in any AX point discovery for that R2ObixClient.

Figure 2-12 shows the property sheet for the **Points** extension of an R2ObixClient, including the default settings for the two “Include” in discovery properties.

Figure 1. R2PointsDeviceExt property sheet with Include defaults

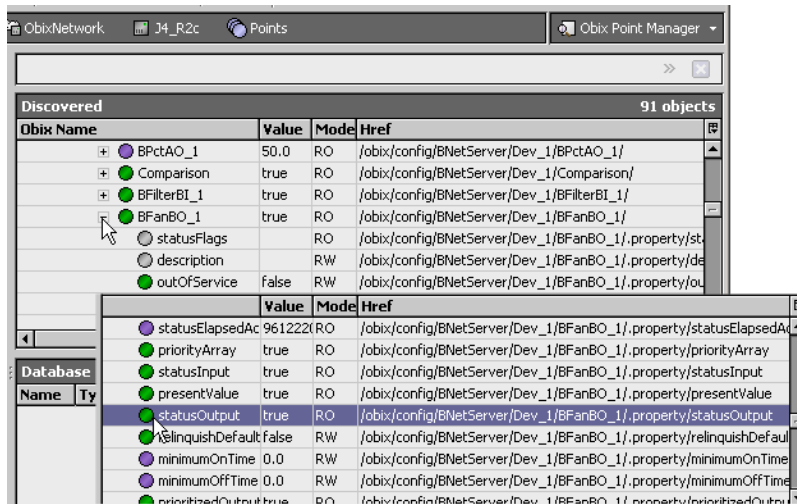


Parent topic: [R2ObixClient Points](#)

Specify the property

When creating proxy points under an R2ObixClient, always select the specific property you want to display, rather than the parent root object (node) itself. For example, expand the entry for a BinaryOutput object and select its statusOutput property (Figure 2-13).

Figure 1. Select property of a discovered R2 object, vs. root (Node) object



Otherwise, if you select the root object rather than a child property, the watch is on the Href for that node. As a result, the watch returns every property of the object. However, only one “default property” is used by the R2ObixClient proxy point— all the other returned properties add overhead, reducing throughput.

Note: Starting in obixDriver builds 3.2.23, 3.3.26, and AX-3.4, proxying any property also provides the parent R2 object’s commands, by default. These appear as actions on each Obix proxy point. In previous obixDriver builds, only “root object” proxy points provided these command actions.

Typical properties selected for proxying/display include “statusOutput”, “prioritizedOutput”, or perhaps “sOut” or “pOut” (varies according to R2 object type). However, note any property is selectable.

In summary, selecting object properties (rather than their root objects) will have a big impact on performance of the oBIX integration. Typically, an order of magnitude or two can be gained by specifying a property for each proxy point, versus specifying the root object.

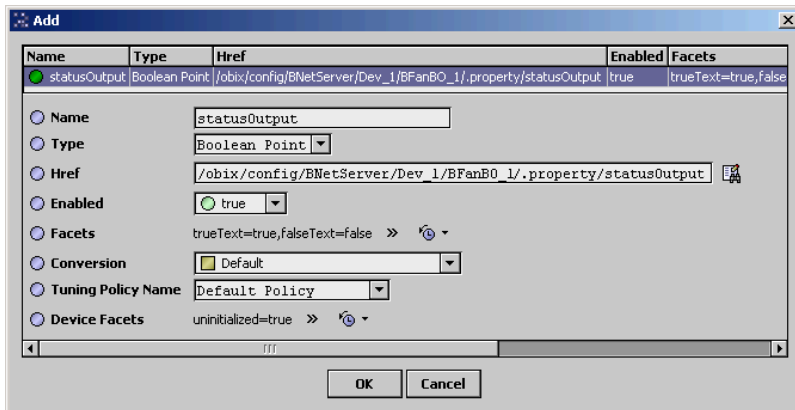
See the next section “Add notes for R2 Obix proxy points” for additional details.

Parent topic: [R2ObixClient Points](#)

Add notes for R2 Obix proxy points

When adding R2 Obix proxy points from the discovered lobby, and selecting a property (recommended), note that currently the default **Name** is the same as the selected R2 property name, such as “statusOutput”, “prioritizedOutput”, and so on. In addition, Facets show default values. See figure below.

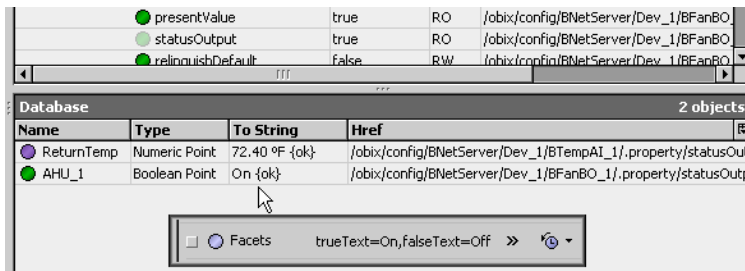
Figure 1. Example Add dialog for R2 Obix proxy point



Edit the default Name to a more descriptive value for that R2 object. If needed, you can find the R2 object's name inside the shown "Href" value (you may need to click inside that field and press End).

Facets in the **Add** dialog do not need editing, even though uninitialized values (sometimes "null") are shown. Upon adding the proxy point to the station, the units (or activeInactiveText) in the source R2 object are automatically uploaded and used as the Facets in the proxy point, as shown below.

Figure 2. Facets are automatically learned from the R2 object's units or activeInactiveText



See the next section "About Obix proxy points" for additional details.

Parent topic: [R2ObixClient Points](#)

About Obix proxy points

Added Obix proxy points often resemble Niagara proxy points in the following ways:

- Most proxy points for R2 objects, including ones for "input writable" ones such as BinaryOutput, AnalogOutput, MultistateOutput, Loop, and so forth are proxied as read-only points only—writable AX point types like BooleanWritable, NumericWritable, etc. are not selectable. This applies to all nodes and properties that appear in the Discovered table with a "Mode" of "RO" (read only).

However, object commands are available, as actions of the read-only proxy point. See figure below and also the "Command Notes" topic.

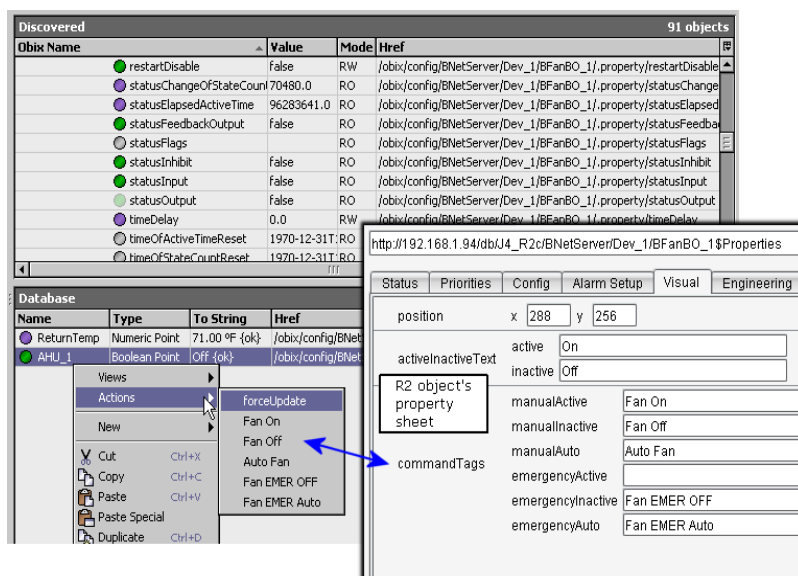
- Some R2 object properties show a discovered "Mode" of "RW". Most are configuration types such as alarm limits (e.g. "lowLimit", "highLimit"), "deadband", and "notificationClass" as a few examples. If desired, you can proxy such a property as a writable point type, e.g. NumericWritable, BooleanWritable, etc. This allows you to write the R2 property value from the AX station, either by an invoked action on the proxy point, or by linking to the proxy point's input(s).

Note: By default, right-click actions on a writable Obix proxy point include both native AX actions for the writable point - to change the value of the specific R2 property actions for the commands on the parent R2 object - to directly command that object

In some cases, you may wish to hide some action slots, e.g. ones for the parent object's commands. Or, if link control (via proxy input) is the only intended method, you may wish to hide all action slots. If hiding any actions, be sure to hide the "forceUpdate" one.

Typically, "standard" control logic linking from AX to an R2 object requires additional engineering on both sides.

Figure 1. Proxy point for R2 object or property offers actions for commands, even as read-only point



- **About forceUpdate**
By default, every Obix proxy point has an available "forceUpdate" action, an admin-level action that results in an immediate fetch of the property's value. If applicable, the forceUpdate also reflects any R2 configuration changes to the parent object's display-related property ("units", "activeInactiveText", etc.).
- **Command Notes**
In addition to a proxy point "forceUpdate" action, note by default "normal right-click" commands appear on the action menu for Obix proxy points—providing the source R2 object has commands. Actions display mirroring the configured R2 object's "commandTags" property strings, where applicable.
- **Link control into R2 objects**
If you need to link local (AX) station control logic into an R2 object input, do this in a similar way as when working between two AX stations (NiagaraNetwork). In either case, you need to create an additional object in the "target input side" i.e. controlled station, and link its output into the object being controlled.
- **AnalogExport Limit Notes**
Note that in the AX station, when invoking an action on an Obix proxy point for any R2 ObixAnalogExport object that is "limit enabled" (see "R2 Obix Export object properties" on page 2-5) any value invoked that is outside the high/low limit range results in an "Invalid Argument" popup message, as shown in Figure 2-19.
- **ObixClient proxy point tips**
When creating Obix proxy points for an R2 controller under the Points extension, the following tips may be useful:
- **Client polling timing**

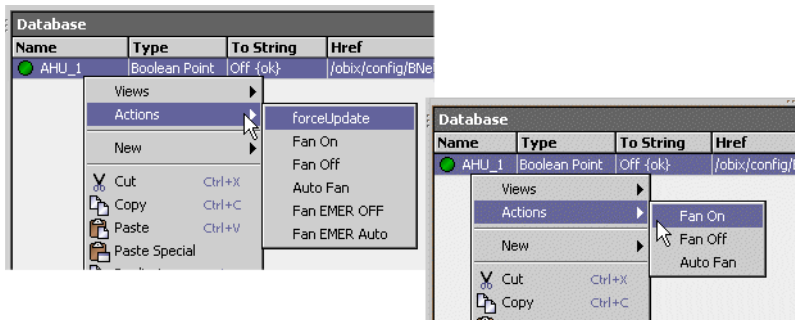
As with most drivers, the polling cycle timing is dependent on the number of proxy points currently in the “Dibs”, “Fast”, “Normal” and “Slow” polls. There are also properties of the R2ObixClient that may need to be adjusted from default values, in support of the oBIX “watch subscription”.

Parent topic: [AxSupervisor engineering](#)

About forceUpdate

By default, every Obix proxy point has an available “forceUpdate” action, an admin-level action that results in an immediate fetch of the property’s value. If applicable, the forceUpdate also reflects any R2 configuration changes to the parent object’s display-related property (“units”, “activeInactiveText”, etc.).

Figure 1. Each Obix proxy point has forceUpdate action available by default



However, sometimes you may wish to hide the forceUpdate action slot, working from the slot sheet view of the Obix proxy point. This is especially true if the proxy point has R2 object command actions that are “hidden,” or have other config flag changes. Otherwise, invoking forceUpdate causes config flag changes to those actions to be overwritten with defaults. For example, any R2 command actions previously engineered to be hidden will display, or command actions set to “Operator” will revert to admin level.

Note: A “global” forceUpdate action is on the **Points** extension of an R2ObixClient. Although seldom exposed on a PxPage, know that it effectively invokes a forceUpdate to all Obix proxy points for an R2ObixClient. Even for just Workbench access, you may wish to hide this action slot (if not already hidden).

Parent topic: [About Obix proxy points](#)

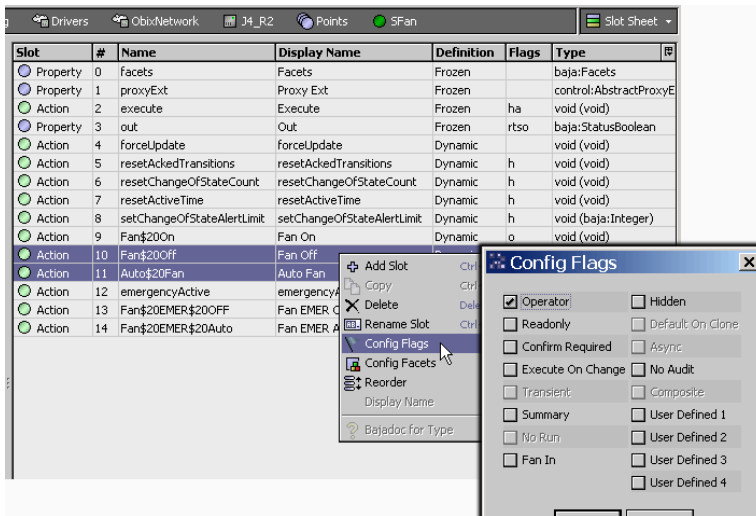
Command Notes

In addition to a proxy point “forceUpdate” action, note by default “normal right-click” commands appear on the action menu for Obix proxy points—providing the source R2 object has commands. Actions display mirroring the configured R2 object’s “commandTags” property strings, where applicable.

If a source R2 control object has “empty” (blank) commandTags properties, note that (by design) corresponding actions do not appear—the same as on the R2 object’s right-click command menu. However, note that those actions do exist on the slot sheet of the Obix proxy point with the Hidden config flag set, by default.

Also by default, note that some additional hidden actions may also exist, depending on the source R2 object type. See “Hidden actions” on page 2-14 for related details.

Note: Actions for R2 object commands that are operator level (“Cmd, Std”) automatically default with the Operator config flag set. See the figure below for how these defaults look from the slot sheet of a proxy point. Figure 1. Obix proxy point from its slot sheet, showing Operator flag set on manual-level actions



Such “manual” (often level 8) commands on R2 objects include:

- manualSet (AnalogOutput, MultistateOutput)
- manualAuto (AnalogOutput, BinaryOutput, MultistateOutput)
- manualActive, manualInactive (BinaryOutput)
- override, cancel, setOverrideValue (AnalogOverride, MultiStateOverride, BinaryOverride)

Note for a few R2 object types, due to the unique “string” content for command names, operator-level actions do not have the operator flag automatically set—for example an Obix proxy point for an R2 “Command” object, or for a BinaryOverride object’s “overrideActive” and “overrideInactive”. If needed, you can explicitly set the operator flags on these actions.

Hidden actions

Other hidden actions may also exist on the Obix proxy point’s slot sheet, including commands normally accessible only on the “Command menu” of the JDE (WorkPlace Pro), when that R2 object’s property sheet is displayed in the JDE for an admin-level user.

Examples of such commands for various control objects include:

- resetAckedTransitions — For all alarm capable objects. Rarely used, it sets any cleared flags in the object’s ackedTransitions property.
- resetChangeOfStateCount — For BI, BO, MSI, MSO objects. Zeroes out any accumulated COS value (numerical changeOfStateCount).
- resetActiveTime — For BI, BO, MSI, MSO objects. Zeroes out any accumulated runtime value (numerical elapsedActiveTime).
- setChangeOfStateAlertLimit — For BI, BO, MSI, MSO objects. To change the numerical COS limit for generating an alert (changeOfStateAlertLimit).
- resetTotal — For a Totalizer object. Zeroes out any accumulated statusTotal value.
- resetCounter — For an NdioHighSpeedCounterInput object. Zeroes out any accumulated totalOutput

and countOutput.

If needed, you can expose any of these type actions by going to the slot sheet of the Obix proxy point, and clearing the “Hidden” config flag. These hidden actions appear listed with an “h” in the Flags column. However, it is anticipated that typically such actions will be left hidden. Note if hiding or unhiding command actions, you should hide the forceUpdate action too.

Parent topic: [About Obix proxy points](#)

Link control into R2 objects

If you need to link local (AX) station control logic into an R2 object input, do this in a similar way as when working between two AX stations (NiagaraNetwork). In either case, you need to create an additional object in the “target input side” i.e. controlled station, and link its output into the object being controlled.

- In a remote AX station, you do this by making a reciprocal Niagara proxy point looking back at the “controlling” (output) point in the local station. See “Link control and Niagara proxy points” in the *Niagara Drivers Guide* for details.
- In the case of the R2 station, you copy one of the three types of “export objects” from the R2 tridiumx/obix jar (ObixAnalogExport, ObixBinaryExport, ObixMultistateExport) and paste it into the station, linking its output into the priorityArray input of the R2 object being controlled. Another link is also required, from the statusOutput of the controlled object back to the feedbackValue input (fln) of the export object. See “R2 ObixExport objects” on page 2-4 for more details.

Then, back in the AX station with the ObixClient, you rediscover the Points (“lobby” object tree), and add a new proxy point for the “root node” of each added export object, selecting the default writable point (NumericWritable, BooleanWritable, etc.) for each one.

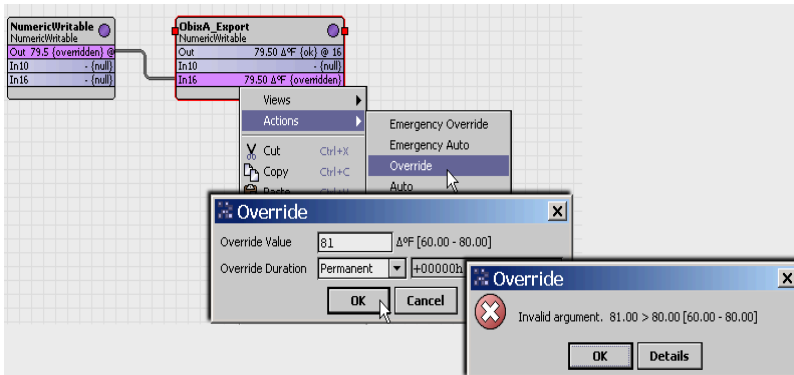
These Obix proxy points provide the array of priority inputs for link control from the local AX station, as well as actions to invoke. Note that as with other discovered R2 objects, in addition to the “root node” they are expandable to select properties to proxy separately. Several config properties may offer utility to proxy as writable types, for instance the “limit” associated properties of ObixAnalogExport objects, providing they are “limit enabled.” See the next section “AnalogExport Limit Notes” for related details.

Parent topic: [About Obix proxy points](#)

AnalogExport Limit Notes

Note that in the AX station, when invoking an action on an Obix proxy point for any R2 ObixAnalogExport object that is “limit enabled” (see “R2 Obix Export object properties” on page 2-5) any value invoked that is outside the high/low limit range results in an “Invalid Argument” popup message, as shown in Figure 2-19.

Figure 1. Invalid Argument popup error in AX Workbench if invoking action outside of limit range.



In addition, be aware that if the active (highest priority) value at the inputs of the NumericWritable Obix proxy point is outside of the limit range established in the corresponding R2 ObixAnalogExport object, this causes the proxy point to go into a fault state, with an updated “Fault Cause” message similar to:

```
Write fault: obix.net.ErrException: <err href="/obix/config/almTest/ExportTest/ObixA_Export/.write"/>
```

Parent topic: [About Obix proxy points](#)

ObixClient proxy point tips

When creating Obix proxy points for an R2 controller under the Points extension, the following tips may be useful:

- As needed, use the column sort features when traversing the lobby tree in the Discovered pane. This can save scrolling time.
- When creating ObixPointFolders for organizing proxy points (double-clicking R2 container type objects, or using the **New Folder** button), be aware of the current database level. Each ObixPointFolder has its own Obix Point Manager view, seen when you double-click it—note this collapses the lobby tree in the discovered pane. However, you can simply re-expand the lobby to find and add child Obix proxy points.
- For any R2 object for which you need standard AX alarming, create a root-level proxy point for it, then add the AX alarm extension to it (for example, an OutOfRangeAlarmExt, duplicating the same alarm and fault (min/maxPresentValue) limits as in the R2 source object). Or, to avoid limits duplication in AX, you can add a StatusAlarmExt and simply select the alarm states in the algorithms for OffNormal and Fault.

Note: Alternatively, you can configure the Alarms device extension under the R2ObixClient to process R2 native alarms within the AX station’s alarm subsystem. This works best for R2 objects that are already being proxied in the AX station. For more details, see “R2ObixClient Alarms” on page 2-18.

- Understand the “forceUpdate” action of an Obix proxy point can cause issues in certain cases. For details, see “About forceUpdate” on page 2-13.

Parent topic: [About Obix proxy points](#)

Client polling timing

As with most drivers, the polling cycle timing is dependent on the number of proxy points currently in the “Dibs”, “Fast”, “Normal” and “Slow” polls. There are also properties of the R2ObixClient that may need to be adjusted from default values, in support of the oBIX “watch subscription”.

Watch Interval

An R2ObixClient device's Watch Interval property controls the polling of the watch, and is found on its **Points** extension. The default watch interval value is 2 seconds. Typically on an R2ObixClient device, the watch interval should be increased to 10 seconds or more, to reduce the load on the R2 platform.

Watch Safety Factor

The parent R2ObixClient also has two related properties, as follows:

- **Watch Safety Factor** — Often, you should also increase the watch safety factor (default is 10 seconds). This specifies the time added to the watch interval to calculate the requested "lease time" of the watch. Essentially, this is the "COV subscription lifetime" the AX client is requesting from the R2 server. This is more important if using a shorter watch interval, because the driver calculates two values to use as the requested lease time, choosing the larger of the two:

```
T1 = watch interval * 2
T2 = watch interval + watch safety factor
```

For large watch intervals, the requested lease time is inherently big (2x). But the safety factor provides an independent way to control requested lease time, such that you could make it 3 or 4 times the watch interval, if desired.

- **Session Timeout** — How long the NiagaraAX client waits on a particular request before giving up. If the R2 station takes longer than 15 seconds to return the watch once polled, you may need to increase the session timeout.

Note: The sessionTimeout slot on an R2ObixClient device is hidden by default. Go to the device's slot sheet and remove the hidden flag from the slot. Once visible, you can go to the property sheet of the R2ObixClient and adjust upward to tune, if needed.

Debug notes

Enabling debug on the R2 server can be useful to diagnose initial problems, but should never be used unless necessary, as it greatly slows the server's response time to the AX Client.

If you encounter errors similar to:

```
HTTP Error 503:Service Unavailable
```

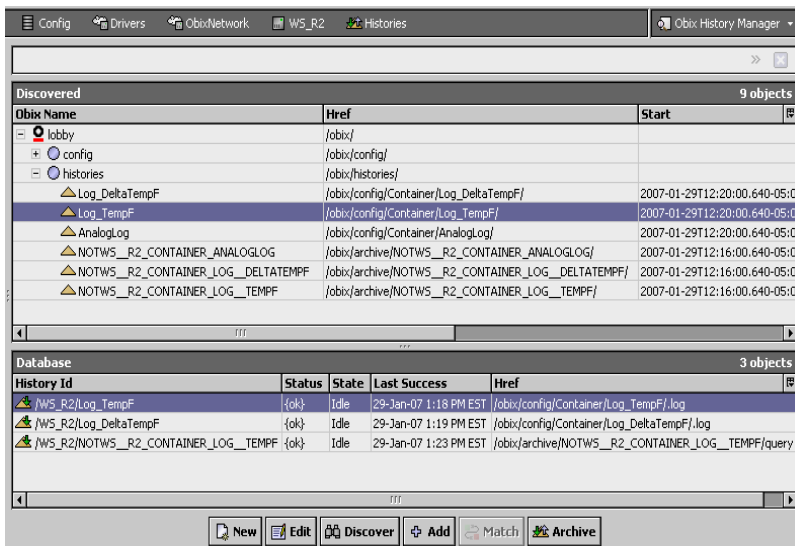
this indicates the R2 host has run out of webservice threads.

Parent topic: [About Obix proxy points](#)

R2ObixClient Histories (logs and archives)

The Histories extension of an R2ObixClient is where you can import data from the R2 station's log objects, as well as existing archives from its appdb (if it has the DatabaseService), by adding history import descriptors. The default view of Histories is the ObixHistoryManager (Figure 2-20), where you specify which discovered logs and archives you want to import, using the familiar oBIX "lobby" tree in the Discovered pane.

Figure 1. ObixHistoryManager is for importing R2 logs and/or archives into the history space



Note: The ObixHistoryManager and created history import descriptors operate as they do in the history import views and descriptors in the NiagaraNetwork and BacnetNetwork. For related details, see the *NiagaraAX Drivers Guide* sections “About the Histories extension” and “History Import Manager”.

Each added descriptor (after archived in AX) produces one history in the station’s local history space, organized by default under a container with the same name as the source R2ObixClient. The following figure shows an example of how the created histories appear in the AX station’s history space.

Figure 2. By default, imported histories under container with name of ObixClient (often, stationName).

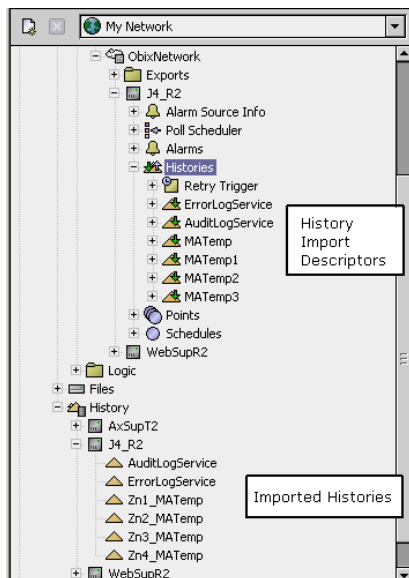


Figure 2-22 shows an example add/edit dialog for an ObixHistoryImport descriptor with default values, where the import descriptor name matches the source R2 Log object name, and the history ID is a combination of the R2ObixClient’s name / Log object name.

Figure 3. Add/Edit dialog for importing an R2 log or archive by import descriptor

| Name | History Id | Execution Time | Enabled | Capacity | Full Policy | Href |
|-----------|------------------|--|---------|-----------|-------------|----------|
| Log_TempF | /WS_R2/Log_TempF | 2:00 AM {Sun Mon Tue Wed Thu Fri Sat } | true | Unlimited | Roll | /obix/co |

☐ Name: Log_TempF
☐ History Id: /WS_R2 / Log_TempF
☐ 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
☐ Href: /obix/config/Container/Log_TempF/.log

Note that an R2ObixClient histories discover includes an R2 station's "AuditLogService" and "ErrorLogService," in addition to logs created by Log objects like "AnalogLog, BinaryLog," and so on. See the next section "ObixClient history import notes" for additional notes.

- **ObixClient history import notes**

When importing logs and archives from the R2 station under the Histories extension, note the following:

Parent topic: [AxSupervisor engineering](#)

ObixClient history import notes

When importing logs and archives from the R2 station under the Histories extension, note the following:

- After you click **Discover** in the **Obix History Manager** to see the **lobby**, your subsequent initial expansion of the "histories" node in the Discovered pane may take a long time to process, often several minutes, depending on the number of log objects in the source R2 station, and particularly how many archives are in a source R2 Web Supervisor station. During this period, the Workbench cursor changes to an "hourglass," and other operations must wait. However, after this initial expansion, the discovered history tree remains cached in memory—at least until you leave the Obix History Manager view.

Note: In a few cases involving large numbers of logs or archives, after expanding the histories node, the Workbench connection to the station was found to timeout and drop. Contact Systems Engineering for assistance in this scenario.

- Note that all log objects and archives appear in the Discover pane after a discover—including log objects with identical names. However, note by default that they are unique by swid/href because of varying locations. It is recommended that you sort (click) the "Obix Name" column in the Discover pane to ASCII-sort discovered logs by name. This will group any identically-named log objects together.

Although the Obix History Import manager allows you to create multiple import descriptors (with default values) for an identical Obix Name, note that only one can successfully import using the same History Id. Import descriptors with a duplicate History Id will go into fault upon import attempt. Therefore, by grouping you can select and edit History Ids appropriately when you add them to the database.

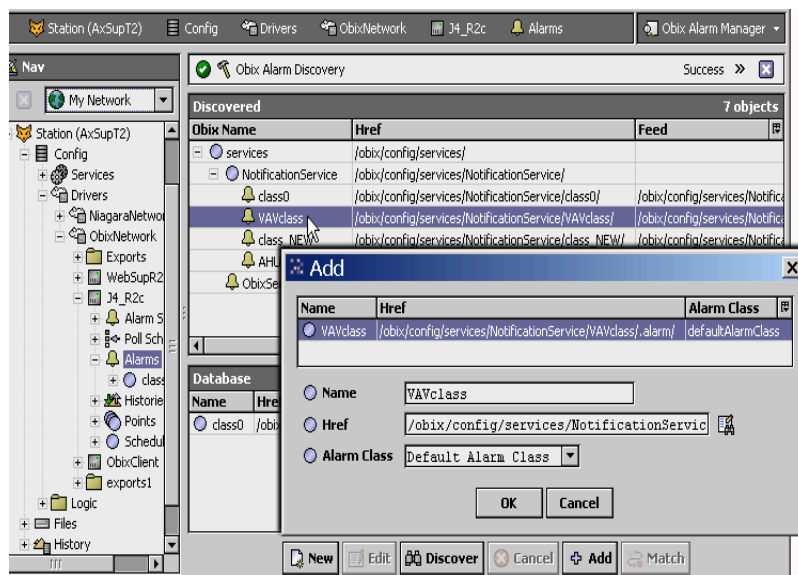
For example, if you had a station named “RN_Hall”, with several logs each named “RmTemp”, you could edit the second field of the History Id for each descriptor to make each unique, for example “Zn1_RmTemp”, “Zn2_RmTemp”, and so on. This way, complete History Ids for each would be “RN_Hall/Zn1_RmTemp”, “RN_Hall/Zn2_RmTemp”, and so forth.

Parent topic: [R2ObixClient Histories \(logs and archives\)](#)

R2ObixClient Alarms

The Alarms extension of an R2ObixClient for the R2 station allows you to add “alarm feed” sources, such that native R2 events (alarms and alerts) in the station can be visible in the AX station’s AlarmService.

Figure 1. Obix Alarm Manager view used to add alarm feeds from R2 NotificationClass objects



To configure, double-click the R2ObixClient’s **Alarms** extension, and in the Obix Alarm Manager view, perform a **Discover**. Expand the **services** node, and then the **NotificationService** node. As shown above, each R2 NotificationClass object is represented as a separate alarm feed, in addition to a “global” “ObixService” alarm feed.

Double-click any feed for the **Add** dialog. If desired, you can select a non-default local Alarm Class, but other properties are typically left at default.

Note: If your AX station has an Alarm Class with the same AX name (as the discovered R2 NotificationClass node), then all imported R2 alarms using that NotificationClass are automatically routed to that local Alarm Class—this overrides any Alarm Class specified in the **Add** dialog.

Click **OK** to add the ObixAlarmImport descriptor(s). These are the only objects added in this view, and they requires no further configuration.

Note: Typically, you add all nodes discovered under the NotificationService, but not the node for the single discovered “ObixService”. This provides the ability for mapping different R2 NotificationClasses to different AX AlarmClasses, rather than just “lumping” all native R2 alarms from the R2 station into a single feed with only one designated AX Alarm Class.

See the following subsections for additional R2 alarm import details:

- R2 alarm import operation
- Example R2 alarm imports
- Final notes on imported R2 alarms
- [R2 alarm import operation](#)
The first thing to understand about importing R2 native alarms is that they are not oBIX “StatefulAlarms,” meaning there is no defined “lifecycle” of an alarm event (unlike with the AX alarming subsystem). However, R2 alarms do support the oBIX “AckAlarm” contract, allowing acknowledgment from AX.
- [Example R2 alarm imports](#)
As previously noted, all imported R2 alarms and alerts from any R2 station appear as “normal” source state events. Also, it is possible that alarms from different R2 source objects will appear under a single row in the Alarm Console (if they are not mapped as Obix proxy points). If you double-click to investigate, you see that they also appear as “normal.” As needed, double-click rows again for the final Alarm Record dialog. See the following figures.
- [Final notes on imported R2 alarms](#)

Parent topic: [AxSupervisor engineering](#)

R2 alarm import operation

The first thing to understand about importing R2 native alarms is that they are not oBIX “StatefulAlarms,” meaning there is no defined “lifecycle” of an alarm event (unlike with the AX alarming subsystem). However, R2 alarms do support the oBIX “AckAlarm” contract, allowing acknowledgment from AX.

Thus, as shown below, all R2 native alarms appear as “normal” (green alarm bell) source state events when routed to the AX Alarm Console. However, all of the “metadata” about each event, including the alarm state (normal, offnormal, etc.), exceeded value, and so on, is included in the alarm details of the alarm record.

Figure 1. Imported R2 alarms always have “Normal” source state in Alarm Console

Alarm Console

Open Alarm Sources

5 Sources / 17 Alarms

| Timestamp | Source State | Ack State | Source | Alarm Class | Priority | Msg Text | |
|---------------------------|--------------|---------------------|-----------------|-------------|----------|-------------------|--|
| 18-Apr-07 12:49:43 PM EDT | Normal | 0 Acked / 3 Unacked | Zone1_T | class0 | 255 | Temp OUT OF RA | |
| 18-Apr-07 12:49:41 PM EDT | Normal | 0 Acked / 7 Unacked | J4_R2c-class0 | class0 | 255 | Humidity out of s | |
| 18-Apr-07 12:48:13 PM EDT | Normal | 0 Acked / 2 Unacked | Zone3_T | class0 | 255 | | |
| 18-Apr-07 12:48:09 PM EDT | Normal | 0 Acked / 4 Unacked | RA_Temp_3 | class_NEW | 255 | | |
| 18-Apr-07 12:40:49 PM EDT | Normal | 0 Acked / 1 Unacked | J4_R2c-VAVclass | VAVclass | 255 | Zone Temp is out | |

fff

Acknowledge

Hyperlink

Notes

Silence

Filter

Alarms are differentiated by alarm feed sources, which typically correspond to different NotificationClasses in the source R2 station (unless, for some reason you choose to only add the “ObixService” as a single alarm feed source). By mapping NotificationClasses to AX Alarm Classes, you can implement similar alarm routing techniques as used in the R2 station—and perhaps more, given that multiple Alarm Consoles can be added/linked to classes. See the next section “R2 alarm source determination” for further details.

R2 alarm source determination

When R2 native alarms from the Obix driver are received in the AX Alarm Console, they display a “Source” string as follows:

- “R2ObixClientName-obixProxyPointName” (if the proxy point for the source R2 object exists in the station database). By convention, this equates to the source R2 station name-proxy point name.
- If the alarm source R2 object is not proxied in the station, then “R2ObixClientName-alarmClassName”.

Therefore, native R2 alarms received that were generated by objects not represented with Obix proxy points will be grouped under a single row (alarm class) in the Alarm Console, and it will not be immediately apparent about the original source object(s) responsible. To investigate further, you must double-click that row, then double-click rows again for the final Alarm Details dialog. See “Example R2 alarm imports” on page 2-19.

Note: In this case especially, you must be careful about acknowledgement of a single row in the Alarm Console, as it may represent several different R2 alarm sources—the single acknowledgement will be applied to all underlying alarms (whether you are aware of them or not).

Be sure to look at the **Ack State** column to see how many unacknowledged alarms exist!

Parent topic: [R2ObixClient Alarms](#)

Example R2 alarm imports

As previously noted, all imported R2 alarms and alerts from any R2 station appear as “normal” source state events. Also, it is possible that alarms from different R2 source objects will appear under a single row in the Alarm Console (if they are not mapped as Obix proxy points). If you double-click to investigate, you see that they also appear as “normal.” As needed, double-click rows again for the final Alarm Record dialog. See the following figures.

Figure 1. R2 alarms by objects not mapped as proxy points require Alarm Details inspection

The screenshot shows the Alarm Console interface with a table of alarms. The table has columns: Timestamp, Source State, Ack State, Source, Alarm Class, Priority, and Msg. The first row shows an alarm from 18-Apr-07 1:25:23 PM EDT, Source State: Normal, Ack State: 0 Acked / 9 Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg: Temp OUT. The second row shows an alarm from 18-Apr-07 12:49:43 PM EDT, Source State: Normal, Ack State: 0 Acked / 3 Unacked, Source: Zone1_T, Alarm Class: class0, Priority: 255, and Msg: Temp OUT. The third row shows an alarm from 18-Apr-07 12:48:13 PM EDT, Source State: Normal, Ack State: 0 Acked / 2 Unacked, Source: Zone3_T, Alarm Class: class0, Priority: 255, and Msg: Temp OUT. The fourth row shows an alarm from 18-Apr-07 12:48:09 PM EDT, Source State: Normal, Ack State: 0 Acked / 4 Unacked, Source: RA_Temp_3, Alarm Class: class_NEW, Priority: 255, and Msg: Temp OUT. The fifth row shows an alarm from 18-Apr-07 12:40:49 PM EDT, Source State: Normal, Ack State: 0 Acked / 1 Unacked, Source: J4_R2c-VAVclass, Alarm Class: VAVclass, Priority: 255, and Msg: Zone Tem.

The Alarm Details dialog is open, showing a detailed view of a specific alarm. The dialog has a title bar with the path: slot:/Drivers/ObixNetwork/J4_R2c/alarms/class0. The dialog contains a table with columns: Timestamp, Source State, Ack State, Source, Alarm Class, Priority, and Msg Text. The first row shows an alarm from 18-Apr-07 1:25:23 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Temp OUT OF RANGE. The second row shows an alarm from 18-Apr-07 1:25:21 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Filter dirty. The third row shows an alarm from 18-Apr-07 12:49:41 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec. The fourth row shows an alarm from 18-Apr-07 12:45:27 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec. The fifth row shows an alarm from 18-Apr-07 12:32:06 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: stationBoot Down @ 0:32 1. The sixth row shows an alarm from 18-Apr-07 12:32:05 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec. The seventh row shows an alarm from 18-Apr-07 12:32:05 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec. The eighth row shows an alarm from 18-Apr-07 12:32:00 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec. The ninth row shows an alarm from 18-Apr-07 12:32:00 PM EDT, Source State: Normal, Ack State: Unacked, Source: J4_R2c-class0, Alarm Class: class0, Priority: 255, and Msg Text: Humidity out of spec.

Alarm Record details show source R2 object by path in href, along with R2 alarm data

The screenshot shows the 'Alarm Record' window with the following fields:

- Timestamp:** 18-Apr-07 1:25:23 PM EDT
- Uuid:** 58e19a6b-3e3d-494e-a1ce-de9246dc7ae1
- Source State:** Normal
- Ack State:** Unacked
- Ack Required:** true
- Source:** slot:/Drivers/ObixNetwork/J4_R2c/alarms/class0
- Alarm Class:** class0
- Priority:** 255
- Normal Time:** null
- Ack Time:** null
- User:** Unknown User

Alarm Data:

- href:** /obix/config/services/NotificationService/class0/.alarm/event/1176917123510/J4_R2c/AlmTest/Zone2_T
- sourceName:** J4_R2c-class0
- msgText:** Temp OUT OF RANGE
- feedName:** class0
- alarmValue:** 89.4
- deadband:** 2.0
- exceededValue:** 85.0
- status:** {inAlarm, unackedAlarm}
- toState:** high_limit
- fromState:** normal
- notifyType:** event
- eventType:** out_of_range
- priority:** 255
- source:** obix:ref
- ackHref:** /obix/config/services/NotificationService/class0/.alarm/event/1176917123510/J4_R2c/AlmTest/Zone2_T
- TimeZone:** America/New_York (-5/-4)

Alarm Transition: Offnormal

Last Update: 18-Apr-07 1:29:48 PM EDT

Buttons: Acknowledge, Hyperlink, Notes, Close

Annotations:

- Ending of href indicates originating R2 object (pointing to the end of the href field)
- Alarm Data fields contain same information in R2 alarm record, such as alarmValue, toState, fromState, and so on (pointing to the status field)

As shown in the figures above, it may be needed to fully expand an imported R2 alarm record to understand its importance. Note that you can use the table options control in the Alarm Console to “Add An Alarm Data Column”, such as “fromState” and “toState”—these may be helpful if you anticipate a lot of R2 alarms like these.

Parent topic: [R2ObixClient Alarms](#)

[Final notes on imported R2 alarms](#)

[Alarm ack differences between R2 and AX](#)

Be aware of the fundamental difference about “alarm ack permissions” between the AX alarming model and the R2 alarming model:

- In the AX system, a station user needs “admin write” permissions on the Alarm Class used to route the alarm in order to acknowledge it, regardless of what permissions (if any) that user may have on the source component that generated the alarm.
- In an R2 system, a station user needs “command, alarm” permissions on the source object that generated the alarm in order to acknowledge it, regardless of what permissions (if any) that user may have on any of the NotificationClass objects.

Keep this in mind when assigning AX user permissions (using categories) to components in the AX station, noting the difference between accessing actions/properties of Obix proxy points vs. acknowledgement of alarms originated from those points.

[Alarm handling discontinued from R2 Web Supervisor](#)

Although the ability to see and acknowledge native r2 alarm/alert events from AX is included (as previously described), the source R2 station (Controller) must have its NotificationService config properties set as follows:

- archiveMode=archive_local_no_SQL
- alarmArchiveAddress, checkbox cleared

This prevents continuation of any R2 Web Supervisor handling of the station’s alarm/alert events.

Possibility of inadvertent acknowledgements

Acknowledgement from AX can occur on individual events (within the popup window for that alarm source) or on all events if **Acknowledge** is pressed while that row is highlighted in the Alarm Console—regardless of how many underlying alarms may exist. This may prove problematic in actual job use. Note this especially applies if many R2 alarm-capable objects are not represented as Obix proxy points in the station.

Alternative to importing native R2 alarms

If alarming is critical, for the reasons previously noted you may wish to transition all alarming to “native AX” alarming, instead of using the ObixClient Alarms feature. Do this by creating Obix proxy points for all objects that require alarming (or runtime/COS count events), and then adding to each point the necessary alarm extension(s), configuring them in AX. Note that extensions for R2 “alert” type events (runtime, COS counts) are found in the alarms folder of the kitControl palette.

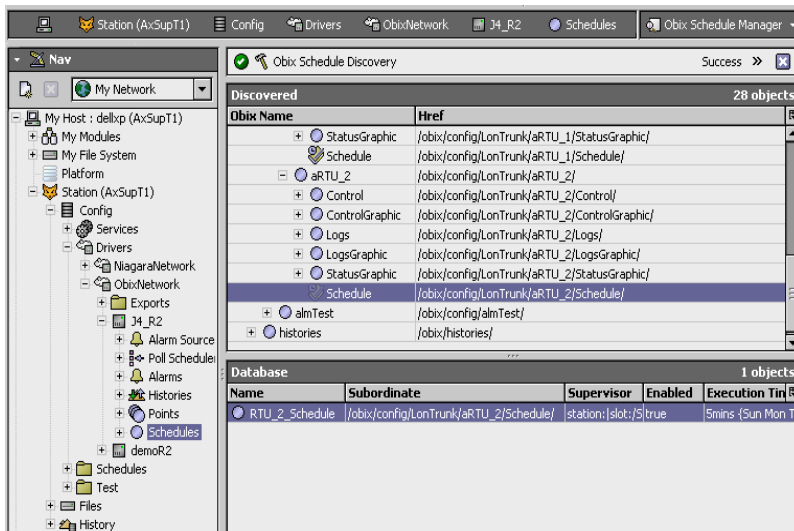
In this “alarm re-engineering” scenario, you would typically create equivalent AlarmClasses for the NotificationClass objects used in the R2 station, configured with similar priorities and alarm recipient components.

Parent topic: [R2ObixClient Alarms](#)

R2ObixClient R2 Schedule exports

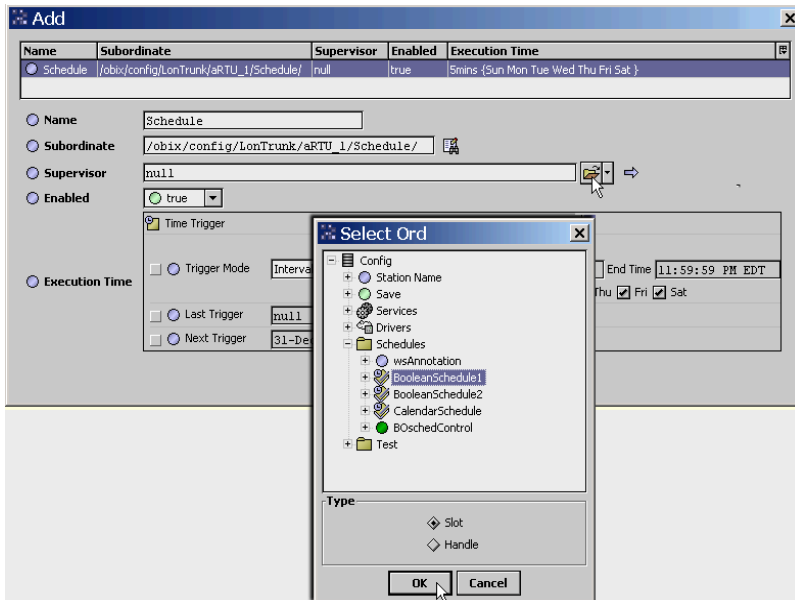
By default, each R2ObixClient has 4 device extensions: Alarms, Histories, Points and Schedules. Expand the R2ObixClient and double-click **Schedules** (R2ScheduleDeviceExt) for the Obix Schedule Manager view, and then perform a **Discover**. Expand the **config** branch of the **lobby** to locate target Schedules in the R2 station. Schedules are added as ObixScheduleExport descriptors, as shown in Figure 2-27.

Figure 1. Obix Schedule Manager view of Schedules device extension to designate R2 slave schedules



In the Add/Edit dialog for a schedule export descriptor, assign component values, including a unique name for the descriptor. In this dialog you must specify which local AX BooleanSchedule will serve as the supervisor (master) schedule for the target R2 schedule. To do this, click the folder icon on the right side of the Supervisor field, which by default opens the **Component Chooser (Select Ord** dialog) as shown in Figure 2-28.

Figure 2. Add/Edit dialog for ObixScheduleExport descriptor, showing Supervisor



In the **Select Ord** dialog of the component chooser, select a schedule and then **OK**. The Supervisor field should now have a valid ord, for example: `station:|slot:/Schedules/BooleanSchedule1`

Add an export descriptor for each R2 Schedule object in the R2 station that you wish to master from the AX BooleanSchedule.

Note: Any target R2 Schedule should not already be “slaved” to an R2 Schedule in another station (typically in the R2 Web Supervisor station), otherwise the AX schedule supervisor function may not be successful. To remove a Schedule from R2 slaved control, delete the externalSubscription link on its “slaveIn” input. A station restart on the slaved station (typically controller) may also be needed.

- **[AX Schedule to R2 Schedule operation](#)**

Although the AX BooleanSchedule and the R2 Schedule seem to have similar “weekly” schedule event programming, they in fact use different “schedule models.” Therefore, AX schedule mastering of an R2 Schedule is implemented by writing the AX schedule’s events to the Special Events in that R2 Schedule.

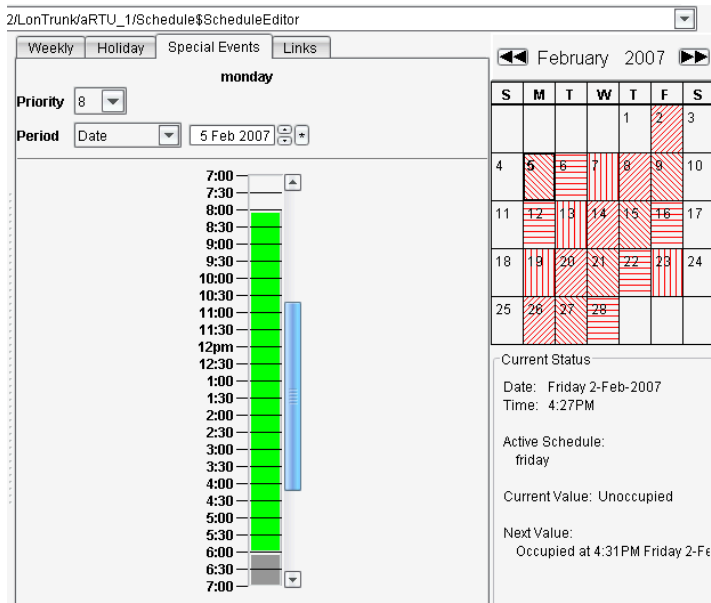
Parent topic: [AxSupervisor engineering](#)

AX Schedule to R2 Schedule operation

Although the AX BooleanSchedule and the R2 Schedule seem to have similar “weekly” schedule event programming, they in fact use different “schedule models.” Therefore, AX schedule mastering of an R2 Schedule is implemented by writing the AX schedule’s events to the Special Events in that R2 Schedule.

Note that unlike AX schedule special events (which “intermingle” with weekly events), R2 schedule special events replace the normal weekly schedule. As shown in below, schedule events that are written to the R2 Schedule from the AX master appear in the “Special Events” tab of the JDE Schedule Editor view, in a block of around four weeks. Each has a “weekday” name (monday, tuesday, monday2, tuesday2, etc.) seen at the top of the tab.

Figure 1. R2 JDE Scheduler Editor view, Special Events tab, showing received AX schedule events



As a contingency measure, you may consider creating additional Schedule objects in the R2 station, along with any necessary additional logic, to provide scheduling control in the case that communications with the AX master station are lost for long periods. Or, continue to maintain the weekly schedule portion of these Schedule objects, because this is not affected by any schedule downloads from the AX Supervisor.

Parent topic: [R2ObixClient R2 Schedule exports](#)

Sample Reports Using BQL and Bound Tables

Customized report graphics are easily created using both a standard Px page or the newer report Px page which was introduced in build 3.2.16. Using the report service and the report pane it is possible to produce more robust interfaces that are easily exported.

The examples presented in this document can be created in older builds with standard Px pages.

- [Creating The Report Px Page](#)

If using the 3.2.x build, there is a new file type which may be added to the station called ReportPxFile.px. This is the preferred default template to use when creating a report graphic. When a standard Px file is created it contains a scroll pane at the root of the widget tree with a canvas pane inside of the scroll pane. The default report Px file only includes a report pane at the root of the widget tree. There is no functional difference between the two pages other than the default panes which are present when the file is created. If using any previous builds older than 3.2.x, then the report pane component is not available.

- [Example Reports](#)

The following sections provide examples of how to create specific types of reports.

Creating The Report Px Page

If using the 3.2.x build, there is a new file type which may be added to the station called ReportPxFile.px. This is the preferred default template to use when creating a report graphic. When a standard Px file is created it contains a scroll pane at the root of the widget tree with a canvas pane inside of the scroll pane. The default report Px file only includes a report pane at the root of the widget tree. There is no functional difference between the two pages other than the default panes which are present when the file is created. If using any previous builds older than 3.2.x, then the report pane component is not available.

- [Report Pane Versus Canvas Pane](#)

The report pane provides several benefits over using a standard Px page with scroll and canvas panes.

- [Using Bound Tables and Bound Labels](#)

You can create a report Px page using bound labels and bound tables to display real time data. Bound tables can also be used to display historical data on the reports as well.

Parent topic: [Sample Reports Using BQL and Bound Tables](#)

Report Pane Versus Canvas Pane

The report pane provides several benefits over using a standard Px page with scroll and canvas panes.

- There are no settings for the report pane size. The report pane is the root of the Px file and auto sizes based on the content to display.
- The report pane has properties which allow the inclusion of a logo image, the page number and a date time stamp on each page of an exported report.
- Bound tables in a report pane auto size to display all rows without the need of a scroll bar, where as a bound table in a standard canvas pane must be sized manually.
- When exporting the report Px to a pdf file the required number of pages automatically generate.

Note: It is important not to put the report pane inside of a scroll pane or the auto sizing and page generation will not function correctly.

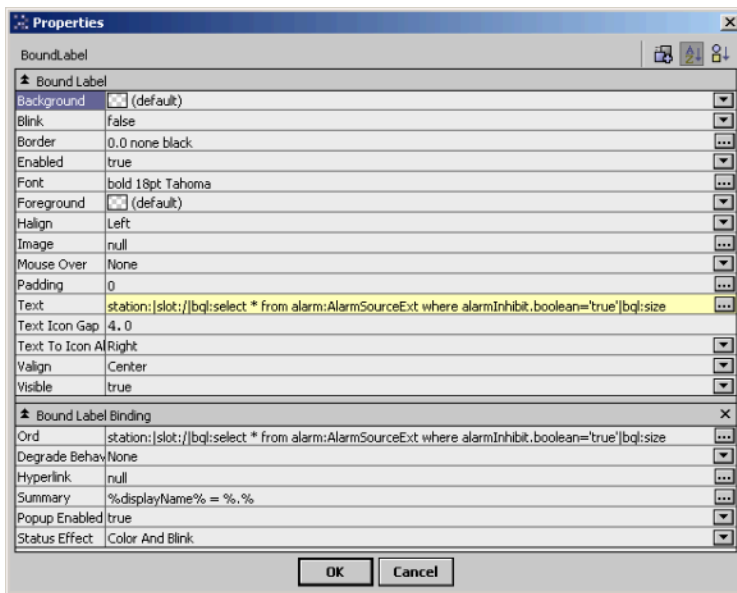
Parent topic: [Creating The Report Px Page](#)

Using Bound Tables and Bound Labels

You can create a report Px page using bound labels and bound tables to display real time data. Bound tables can also be used to display historical data on the reports as well.

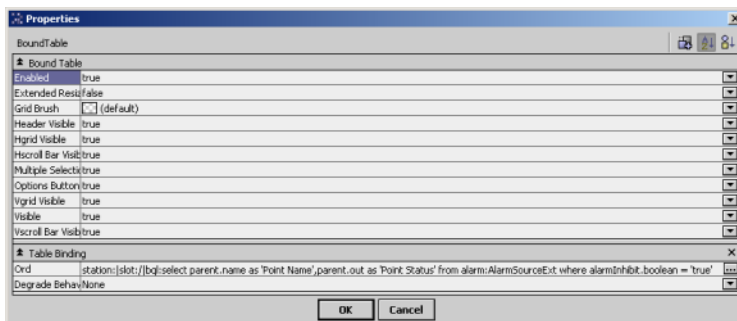
A bql query can be entered in the ord field of the bound label binding (see 6-1), as opposed to referencing a component in the station. Animating the text field allows displaying the results of the bql query. If the bformat text is configured to '%%' then it will simply display the value. Additional standard text may be used as well such as, 'Points With Alarms Disabled (Alarm Inhibit = True): %%'.

Figure 1. Bound label property sheet using bql query in the Ord field



The bql query is entered in the ord field of the bound table to display the list of points or historical data (see 6-2).

Figure 2. Bound table property sheet using bql query in Ord field



Parent topic: [Creating The Report Px Page](#)

Example Reports

The following sections provide examples of how to create specific types of reports.

- Point Status Report
- Schedule Report
- Tenant Override Report
- Weekly Electrical Demand Report
- Point Status Report
- **Point Status Report**
Operators often desire a snap shot report which displays the status of points. In particular emphasis is placed on points which are in an abnormal state such as overridden, alarm, fault, disabled or which have alarm functionality disabled.
- **Schedule Report**
Operators often request a print out of all scheduled events including the normal weekly schedules and special events. A schedule report can be generated for all of the schedules in a given station by using a bql query.
- **Tenant Override Report**
It is often times necessary to generate reports on historical data, which can be equally as important as reporting on real time data. Histories in the station may be configured to record information at specified intervals, a change of value or state, or only when certain other conditions exist. There are also standard logs in the station like the audit log and log history which record operator actions and system errors.
- **Weekly Electrical Demand Report**
Bound tables and charts can be used in a report Px page to display historical information. An example might be to create a report which displays electrical demand readings for the previous week.

Parent topic: [Sample Reports Using BQL and Bound Tables](#)

Point Status Report

Operators often desire a snap shot report which displays the status of points. In particular emphasis is placed on points which are in an abnormal state such as overridden, alarm, fault, disabled or which have alarm functionality disabled.

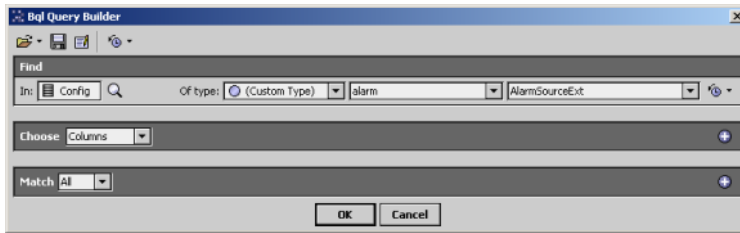
- Displaying Points with Alarm Functionality Disabled

Alarm extensions have a boolean property called 'Alarm Inhibit'. When the property value is 'true', processing of alarm events is prevented. The alarm inhibit status is not displayed in the status of the parent point because it is not actually a property of the BStatus type.

Using bql the station can be searched for components which are alarm source extensions. The bql query builder (see 6-3) can be used to create the query or you can simply type the syntax, as shown below.

```
station:|slot:|/bql:select * from alarm:AlarmSourceExt
```

Figure 1. Bql query builder constructing query for alarm source extensions

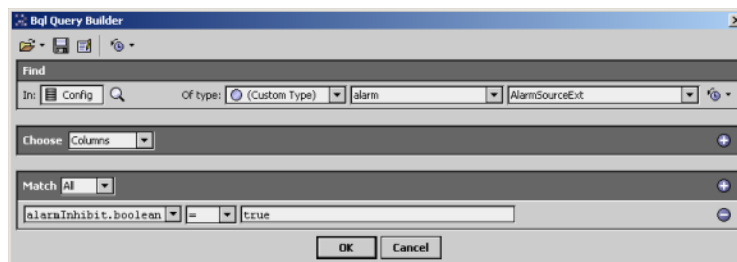


– Filtering by Alarm Inhibit Property

This query returns a list of all alarm source extensions in the station regardless of whether the extension is inhibited or not. To limit the return based on the alarm inhibit property, the query needs to be modified as below (see 6-4).

```
station:|slot:/|bql:select * from alarm:AlarmSourceExt where
alarmInhibit.boolean = 'true'
```

Figure 1. Bql query builder constructing query filtered by alarm inhibit property

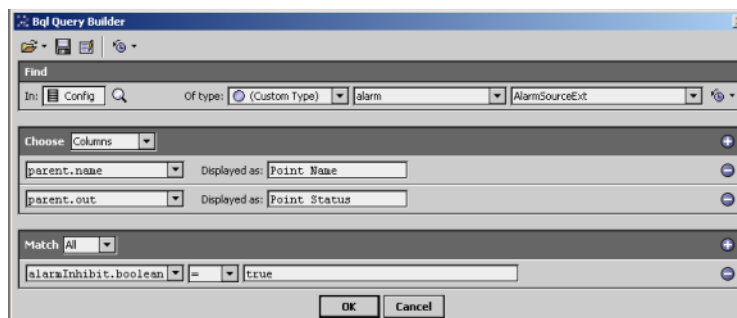


– Displaying Specific Columns

The filtered query returns a list of alarm source extensions whose alarm inhibit property is set to true, but the table displays all of the properties for the extension. It is preferable to further filter the results so that only the desired properties are displayed. The columns which are displayed in the table may be limited by further modifying the query as below (see 6-5).

```
station:|slot:/|bql:select parent.name as 'Point Name',parent.out
as
'Point Status' from alarm:AlarmSourceExt where alarmInhibit.bool
ean = 'true'
```

Figure 2. Bql query builder constructing query to filter the displayed columns



Note: In the example above, bformat text is used to display information from the parent component of the alarm source extension. The name or displayName of the parent point and

the out slot would likely be useful information to display. Other columns could be added to display information from the alarm source extension as well.

- Displaying the Number of Records in a Query

It is also possible to use a bql query to calculate and display the number of records returned in the query. The original query can be modified as below.

```
station:|slot:|/|bql:select * from alarm:AlarmSourceExt where
alarmInhibit.boolean = 'true'|bql:size
```

Note: The bql query builder does not support the '|bql:size' syntax, although it is a valid query. It is necessary to delete this syntax from the query prior to opening the bql query builder.

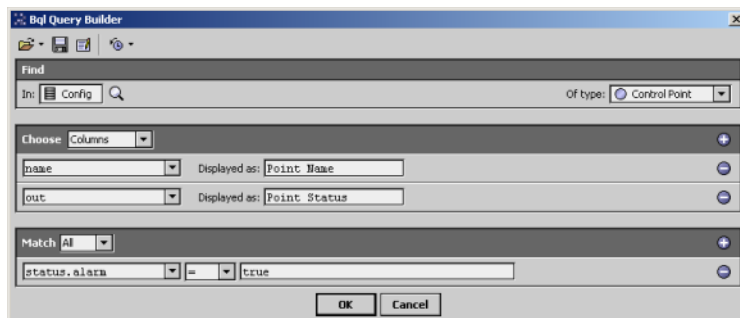
- Displaying Points Currently In Alarm

Each control point in the station has both a value and a status. The status reflects the current condition or reliability of the point value. Available statuses are down, alarm, unacknowledged alarm, overridden, disabled, fault and stale.

Using bql the station can be searched for control points where the alarm status flag is true. The bql query builder (see 6-6) can be used to create the query or you can simply type the syntax below.

```
station:|slot:|/|bql:select name as 'Point Name',out as 'Point Status'
from
control:ControlPoint where status.alarm = 'true'
```

Figure 4. Bql query builder constructing query for control points in alarm

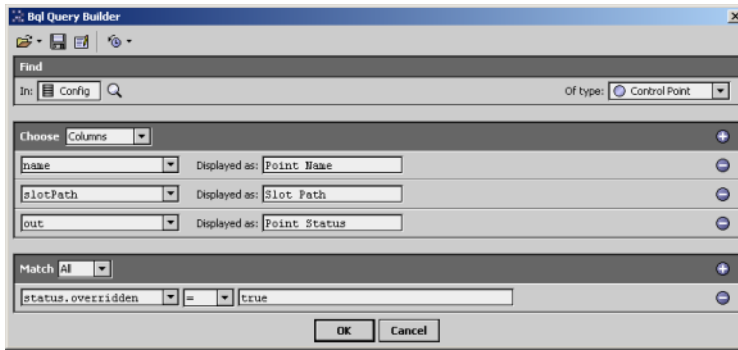


- Displaying Points Currently Overridden

Using bql the station can be searched for control points where the overridden status flag is true. The bql query builder (see 6-7) can be used to create the query or you can simply type the syntax below.

```
station:|slot:|/|bql:select name as 'Point Name',out as 'Point Status'
from
control:ControlPoint where status.overridden = 'true'
```

Figure 5. Bql query builder constructing query for control points currently overridden



Parent topic: [Example Reports](#)

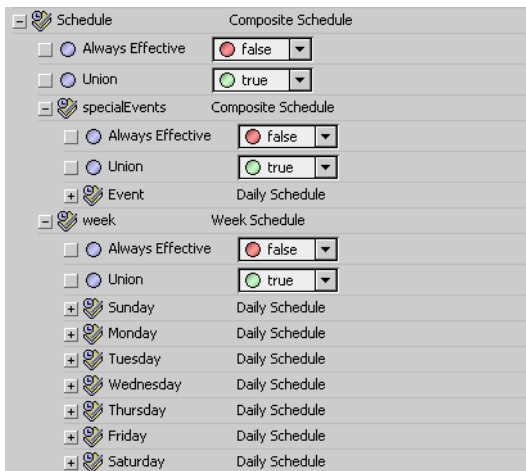
Schedule Report

Operators often request a print out of all scheduled events including the normal weekly schedules and special events. A schedule report can be generated for all of the schedules in a given station by using a bql query.

- Examining a Typical Schedule

By default the composite schedule component of a standard schedule is hidden. Accessing the slot sheet of a schedule component allows removing the hidden flag from the composite schedule. Viewing the composite schedule property sheet (see 6-8) helps to understand how the composite schedule is organized and how to construct the bql query.

Figure 1. Composite schedule property sheet

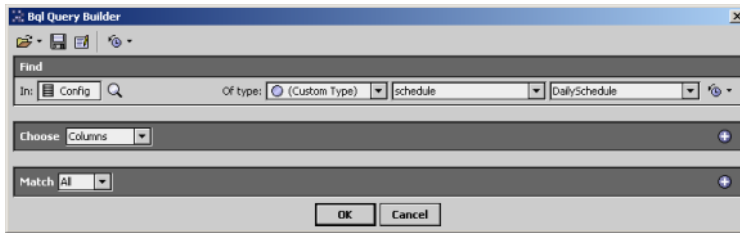


The main composite schedule consists of a child composite schedule called `specialEvents` and a week schedule called `week`. Special events which are added to the schedule and each specific day of the week schedule are all daily schedule type objects.

Using bql the station can be searched for the daily schedule components. The bql query builder (see 6-9) can be used to create the query or you can simply type the syntax below.

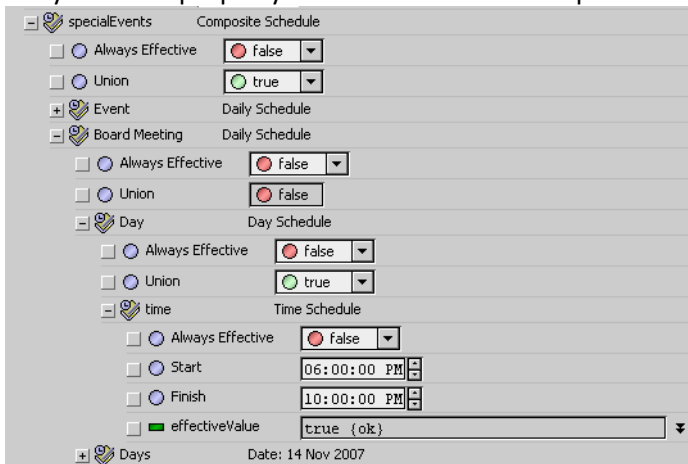
```
station:|slot:|/bql:select * from schedule:DailySchedule
```

Bql query builder creating query for daily schedule components



This basic query for the daily schedule component returns the slot path, property values and names of the sub schedule components. This is not very useful information for an operator, so the query will likely need to be modified to display more specific pertinent information. The property sheet of the daily schedule can be used to get a better idea of what data might be useful to display in the columns (see 6-10).

DailySchedule property sheet with children components expanded



Each daily schedule consists of two child components. The first component is called 'day' which is a day schedule and the second component is called 'days' which is an abstract schedule. The abstract schedule defines the event date and the day schedule defines the event times and event value.

```
Weekly Schedule (Boolean Schedule) %parent.parent.parent.name%
|-- Schedule (Composite Schedule) %parent.parent.name%
|   |-- Special Events (Composite Schedule) %parent.name%
|   |   |-- Board Meeting (Daily Schedule)
|   |   |   |-- Day (Day Schedule)
|   |   |   |   |-- time (Time Schedule)
|   |   |   |   |-- Days (Abstract Schedule)
|   |   |-- week (week schedule)
|   |   |   |-- Sunday (Daily Schedule)
|   |   |   |   |-- Day (Day Schedule)
|   |   |   |   |   |-- time (Time Schedule)
|   |   |   |   |   |-- Days (Abstract Schedule)
```

- Resolving the Schedule Name

Since the bql query is for the daily schedule components, it is necessary to use BFormat text to derive the schedule name using the below syntax. The actual schedule component is three levels up the navigation tree from the daily schedule components.

```
%parent.parent.parent.displayName%
```

- Displaying the Event Date

The days component returns the actual event date if specified or if it is a weekly schedule then it displays 'weekday schedule'.

- Displaying the Event Name

Since the query returns the daily schedules, use 'displayName'.

- Displaying the Event Times

The event times are slots of the time schedule which is a child of the daily schedule components. There are individual slots for the start and finish times, which may be displayed using the below syntax.

```
day.time.start (displays the time of the first start event)
day.time.finish (displays the time of the first stop event)
day.time1.start (displays the time of the second start event)
day.time1.finish (displays the time of the second stop event)
```

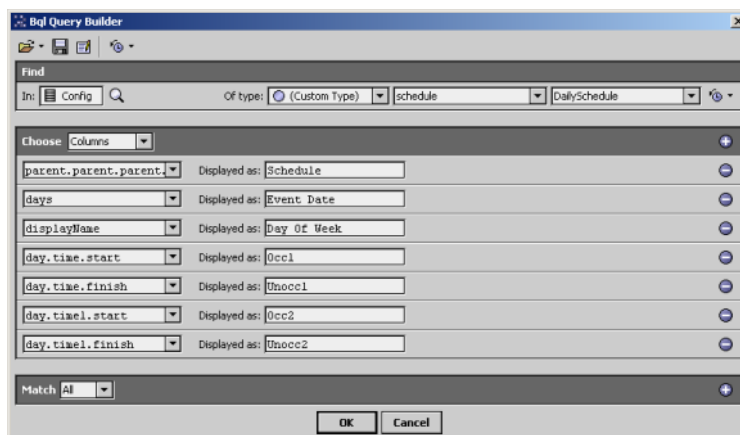
- Filtered Query for Daily Schedules

Using the above concepts a filtered query can be created to display the desired results. The bql query builder (see 6-11) can be used to create the query or you can simply type the syntax below.

```
station:|slot:|/bql:select parent.parent.parent.displayName as 'Schedule'
le',days
as 'Event Date',displayName as 'Day Of Week',day.time.start as 'Occ1',
day.time.finish
as 'Unocc1',day.time1.start as 'Occ2',day.time1.finish as 'Unocc2' from
m
schedule:DailySchedule
```

Each daily schedule consists of two child components. The first component is called 'day' which is a day schedule and the second component is called 'days' which is an abstract schedule. The abstract schedule defines the event date and the day schedule defines the event times and event value.

Figure 2. Bql query builder creating query for daily schedule components



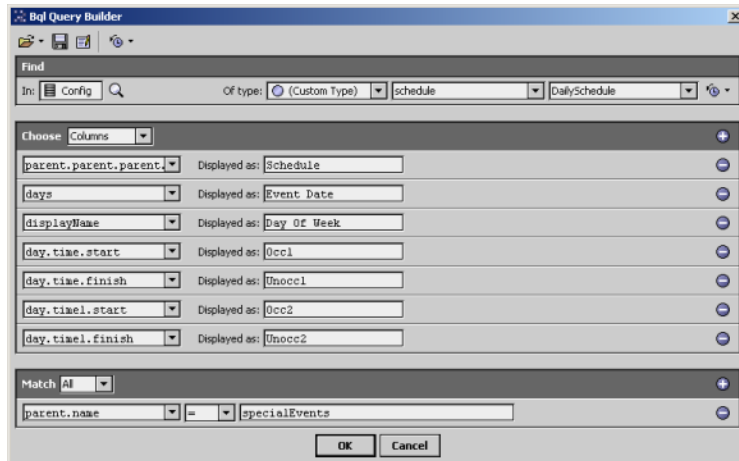
- Filtering by Special Events

It may also be helpful to filter the results to only display special events, after all people typically know what the standard schedule events are. The bql query builder (see 6-12) can be used to modify the

query or simply type the syntax below.

```
station:|slot:|bql:select parent.parent.parent.displayName as 'Schedule',days as 'Event Date',displayName as 'Day Of Week',day.time.start as 'Occ1',day.time.finish as 'Unocc1',day.time1.start as 'Occ2',day.time1.finish as 'Unocc2' from schedule:DailySchedule where parent.name = 'specialEvents'
```

Figure 3. Figure 12: bql query builder creating query for special events



Parent topic: [Example Reports](#)

Tenant Override Report

It is often times necessary to generate reports on historical data, which can be equally as important as reporting on real time data. Histories in the station may be configured to record information at specified intervals, a change of value or state, or only when certain other conditions exist. There are also standard logs in the station like the audit log and log history which record operator actions and system errors.

If a tenant has access to override set points or scheduled periods of operation, then it may be a requirement to document these actions. The standard audit log can be used to track the actions, or additional histories could be configured to track the actions as well.

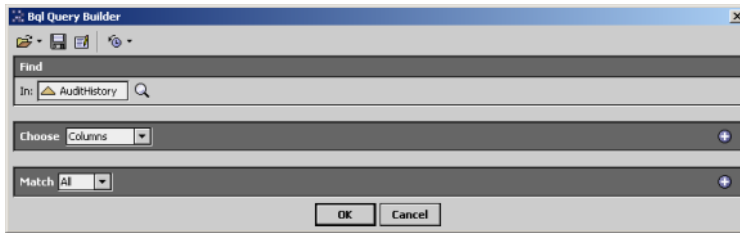
- Using the Audit Log

The audit log can be searched for records where a specific set point or component has been overridden. The audit log record displays the timestamp, operation (added, removed, invoked, changed, etc), target (path to component), slot name, old value (occupied, unoccupied, 78 deg F, etc), value (the new value), and the user who performed the operation.

The bql query builder (see 6-13) can be used to create the query or you can simply type the syntax below.

```
history:/demo/AuditHistory|bql:select *
```

Figure 1. Bql query builder creating query for the audit log

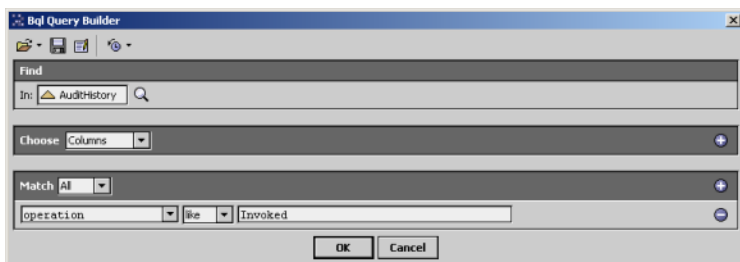


- Filtering by Operation

This basic query returns a list of every record in the audit log and every available data column. Since the objective is to only display records relating to an overridden point, the query can be modified as below (see 6-14) to limit the return based on the operation

```
history:/demo/AuditHistory|bql:select * where operation like 'Invoked'
```

Figure 2. Figure 14: bql query builder creating query for the audit log filtered by operation

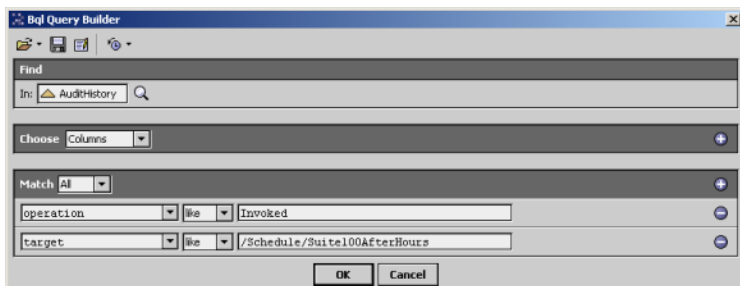


- Filtering by Target Component

The modified query only returns records where the operation is 'invoked'; however, the return is for every component in the station. The query can be further modified as below to limit the return based on the (target) component. (see 6-15)

```
history:/demo/AuditHistory|bql:select * where operation like 'Invoked'
and target
like '/Schedule/Suite100AfterHours'
```

Figure 3. Bql query builder creating query for the audit log filtered by operation and target



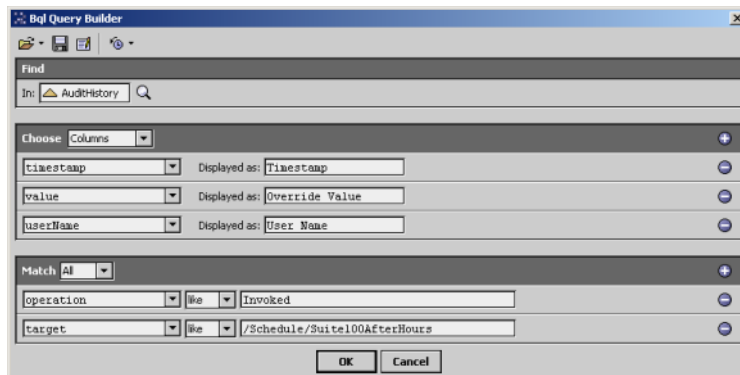
- Configuring the Displayed Columns

The return could be cleaned up by further modifying the query as below (see 6-16) to limit the displayed columns.

```
history:/demo/AuditHistory|bql:select timestamp as 'Timestamp',value a
```

```
s 'Override Value',
userName as 'User Name' where operation like 'Invoked' and target like
'/Schedule/Suite100AfterHours'
```

Figure 4. Creating query for the audit log filtered by operation, target and columns

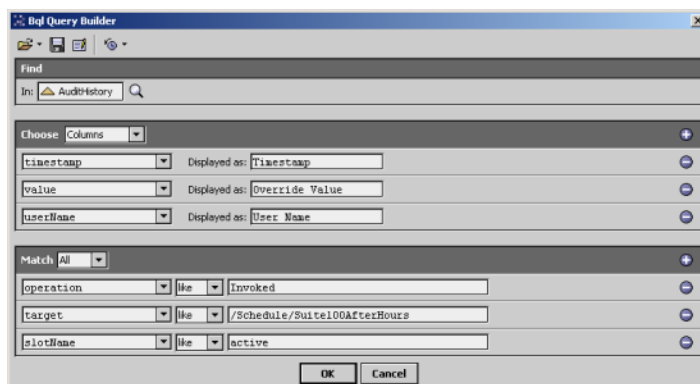


- Filtering by Specific Actions

The return is now limited to actions which have been invoked on a specific component in the station by any operator. It may be necessary to further filter the results to a single operator or a specific action. The bql query can be modified as below (see 6-17) to limit the return to the specific operation of operator override to the active state.

```
history:/demo/AuditHistory|bql:select timestamp as 'Timestamp',value a
s 'Override Value',
userName as 'User Name' where operation like 'Invoked' and target like
'/Schedule/Suite100AfterHours' and slotName like 'active'
```

Figure 5. Creating query for the audit log filtered by operation, target, columns and slot



- Filtering by Timestamps

The bql query now produces a fairly presentable table which displays the timestamp, value and operator for each record. The return contains records from any time period in the audit log. Depending on how many records are maintained in the audit log, this could be a fairly large list. It is likely that the query will need to be modified as below to limit the return based on desired date ranges.

```
history:/demo/AuditHistory?period=lastMonth|bql:select timestamp as 'T
imestamp',value as
'Override Value',userName as 'User Name' where operation like 'Invoke
```

```
d' and target like
'/Schedule/Suite100AfterHours'
```

Note: The bql query builder does not support the '?period=' syntax, although it is a valid query. It is necessary to delete this syntax from the query prior to opening the bql query builder.

Parent topic: [Example Reports](#)

Weekly Electrical Demand Report

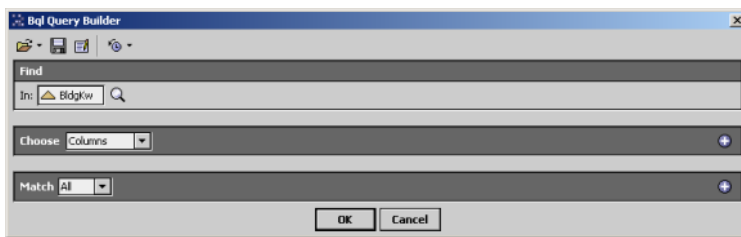
Bound tables and charts can be used in a report Px page to display historical information. An example might be to create a report which displays electrical demand readings for the previous week.

- Creating the History Query

Any history in the station can be queried using bql. The bql query builder can be used to create the query or you can simply type the syntax below.

```
history:/demo/BldgKw|bql:select *
```

Figure 1. Creating query for the BldgKw history



- Limiting the Query

The above bql query will return a table which contains all of the records available in the referenced history. If the export source object is executed once a week, then it would probably be safe to limit the query to the previous weeks worth of data. The period syntax can be used to limit the query as below.

```
history:/demo/BldgKw?period=lastWeek|bql:select *
```

This will still return a significant amount of data. Assuming that the data is recorded in fifteen minute intervals, this would return 672 records and the pdf file would span seventeen pages. Bql includes a special function called history rollup. This function can be applied to the previous query to display the same results using fewer records. Each record in the rollup indicates the number of samples, minimum value, maximum value, average value and a sum of all samples. The below example uses the history rollup function with a daily interval which results in only seven records (one for each day of the week) displayed in the table.

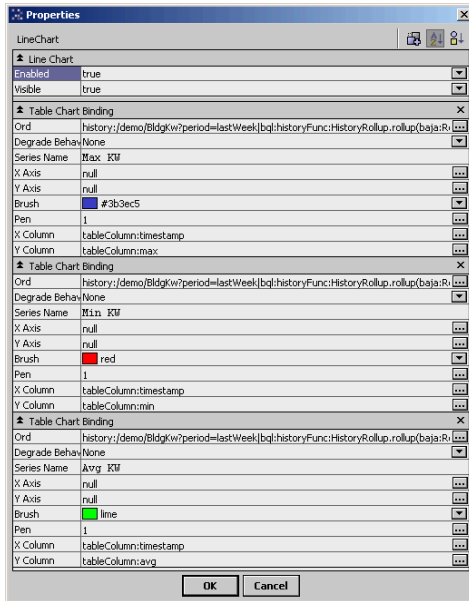
```
history:/demo/BldgKw?period=lastWeek|
bql:history:HistoryRollup.rollup(baja:RelTime '86400000')
```

Note: The history rollup function does not allow specifying which columns are to be displayed. By default all columns are displayed in the table.

- Using a Chart

In some cases it may be a requirement to display the data in chart form as opposed to using a table. The chart palette includes two types of chart widgets, the line and bar chart. After adding the line chart widget to the Px file, table chart bindings may be added and configured. The previous query can be used in the ord field of the table chart binding. Each table chart binding represents one trace on the chart and must specify which column of the query to display on the Y axis. Multiple table chart bindings can reference the same query but display different columns such as min, max, avg or sum.

Figure 2. Creating table chart queries for the BldgKw history



Note: The current export mechanism does support using history chart tables or history chart views directly in the report Px file. Both the history table and chart views allow exporting directly from the specific view.

Parent topic: [Example Reports](#)

Scientific Notation Support

Niagara supports the use of very large and very small numbers by using scientific notation. Values equal to or larger than 9007199254740992 are rendered in scientific notation using "E Notation" format. The following sections use examples to describe how AX displays values in scientific notation:

- E Notation Format
- Example Number Expressions
- [E Notation Format](#)
The general format for scientific notation "E Notation" is: nEx
- [Example Number Expressions](#)
Three types of scientific notation examples follow:

E Notation Format

The general format for scientific notation "E Notation" is: nEx

Where:

n = the "coefficient", a number that is greater than 1 and less than 10^[1]
signifies the exponent place, or base 10^[2]
 x = the exponent (or "power of") 10

Note: You may enter numbers in property fields as either standard expression or as scientific notation. How the numbers are displayed depends on the size of the number.

Parent topic: [Scientific Notation Support](#)

Example Number Expressions

Three types of scientific notation examples follow:

- Example 1: Maximum and Minimum Numbers
- Example 2: Numbers Displaying in a Graphic Display (Px Page)
- Example 3: Numbers Displaying in a Property Sheet View
- [Example 1: Maximum and Minimum Numbers](#)
The following table shows a comparison of how numbers display.
- [Example 2: Numbers Displaying in a Graphic Display \(Px Page\)](#)
The following illustration shows an example of how scientific notation expresses a large number in a graphic display of a process application.
- [Example 3: Numbers Displaying in a Property Sheet View](#)
The following illustration shows an example of how scientific notation expresses a large number in the Workbench property sheet.

Parent topic: [Scientific Notation Support](#)

Example 1: Maximum and Minimum Numbers

The following table shows a comparison of how numbers display.

Scientific Notation Display Comparison

| Example # | Number Used: | Workbench Displays: |
|-----------|-------------------|-----------------------|
| 1. | 9007199254740991 | 9007199254740991 |
| 2. | 9007199254740992 | 9.007199254740992E15 |
| 3. | -9007199254740992 | -9.007199254740992E15 |
| 4. | -9007199254740991 | -9007199254740991 |

Note the following about these example numbers:

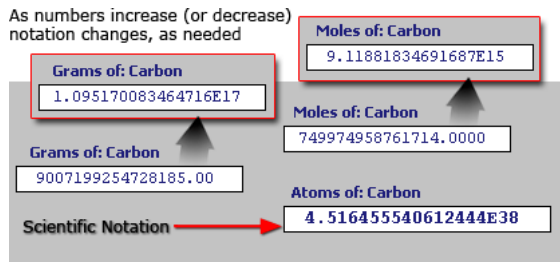
- In Example 1:
the number used (entered or calculated) is the largest number that displays in standard expression (without scientific notation). If "1" is added to this number, then it is displayed as the number in Example 2.
- In Example 2:
the number used (entered or calculated) is displayed using scientific notation.
- In Example 3
the number used (entered or calculated) is displayed using scientific notation.
- In Example 4
the number used (entered or calculated) is the smallest number that displays in standard expression (without scientific notation). If "1" is subtracted from this number, then it is displayed as the number in Example 3.

Parent topic: [Example Number Expressions](#)

Example 2: Numbers Displaying in a Graphic Display (Px Page)

The following illustration shows an example of how scientific notation expresses a large number in a graphic display of a process application.

Figure 1. Displaying Large Numbers Using Scientific Notation



Note the following about Example 2:

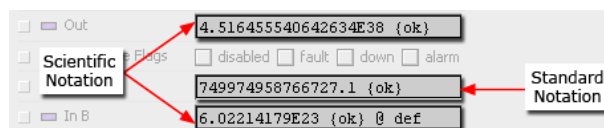
- Numbers are displayed using both scientific notation and standard expression on a graphic Px page.
- As a source amount of carbon exceeds a certain value, the expression of that value in "grams", "moles", and "atoms" of carbon changes from standard expression to scientific notation. If the source value is reduced, the value expressions can change back to standard notation from scientific notation, as well.

Parent topic: [Example Number Expressions](#)

Example 3: Numbers Displaying in a Property Sheet View

The following illustration shows an example of how scientific notation expresses a large number in the Workbench property sheet.

Figure 1. Scientific Notation in the Property Sheet View



Note the following about Example 3:

- Property fields display both standard notation and scientific notation, as needed.
- Facet settings for "Precision" do not limit the number of decimal places in the scientific notation expression. For example, a number that is displayed with a precision of "1" decimal place may display with more than one decimal place in scientific notation.

Parent topic: [Example Number Expressions](#)

BQL Expression component

BQL Expression component (Expr), found in the Util folder of the kitControl palette, is a multi-purpose wire sheet object for mathematical and logical operations that augments the existing math and logic components already present in the kitControl palette.

Expr can reduce the number of component instances in your station by allowing you to quickly and easily create simple logic and math statements. You do not need Java programming knowledge, a Program component, or even multiple standard components to do this. Expr does not require compilation.

Note: This component is available in a station running on the AX-3.6 and later host. Sections in this document describe the component and how to use it.

- [Component features](#)
BQL Expression component supports:
- [What the component is not](#)
BQL Expression component is not:
- [Syntax](#)
The standard syntax for an expression is as follows:
- [Create a BQL Expression component](#)
- [Handling null](#)
This section describes expression handling of a “null” input from a linked status type output.
- [Troubleshooting](#)
If you enter the wrong syntax, or Expr recognizes that you are trying something illegal, it displays status and fault cause information.
- [Frequently-asked questions](#)
How many inputs and outputs does Expr allow?

Component features

BQL Expression component supports:

- Math and logic operators
- Multiple expressions (delimited by commas) within a single component
- Dynamically-created Expr inputs and output(s) based on the expression(s)
- Automatic link conversion between different data types, for example, Double to StatusNumeric

When configuring the component, you specify all of its input slots to “Execute On Change”, such that the Expr executes upon any input change. However, the component also allows you to override this behavior by specifying a time delay between executions. This can be useful to minimize the effects of rapidly changing inputs. Other properties provide status and fault cause information.

In addition to the blank Expr component found in the Util folder of the kitControl palette, two example Expr components: ExprLogic (in the Logic folder), and ExprMath (in the Math folder) each use a single BQL expression to demonstrate how the component works.

Parent topic: [BQL Expression component](#)

What the component is not

BQL Expression component is not:

- A replacement for the Program component. BQL Expression component does not support `//remarks` and `//comments`.
- A replacement for other BQL statement containers that manipulate data.
- Capable of manipulating data through stored variables.
- Capable of line-by-line programming.
- Capable of handling time functions.

If your requirements exceed what can be achieved using a BQL Expression component, there may come a point when you have to consider using a Program component.

Parent topic: [BQL Expression component](#)

Syntax

The standard syntax for an expression is as follows:

input operator 'output'

where:

input is the name of one or more slots.

operator is a word or symbol.

'output' is the slot that contains the result of the expression. (note apostrophes around slot name)

- [Supported operators](#)
- [Commas](#)
If creating a Expr component with multiple expressions (output slots), use a comma to delimit expression statements from one another.
- [Long statements](#)
When entering an expression, keep typing or enter a CR/LF to wrap a long statement.

Parent topic: [BQL Expression component](#)

Supported operators

As Expr uses BQL, all the standard BQL expression syntax is available. For more information on BQL expressions, see the *Niagara Developer Guide*.

Operators are processed by their precedence, that is "order of operation", from first (1) to last (6).

1. `!`, `not`, `-`
logical not, numeric negation
2. `*`, `/`
multiplication, division
3. `+`, `-`

addition, subtraction

4. =, !=, >, >=, < <=, like

comparisons

5. and, or

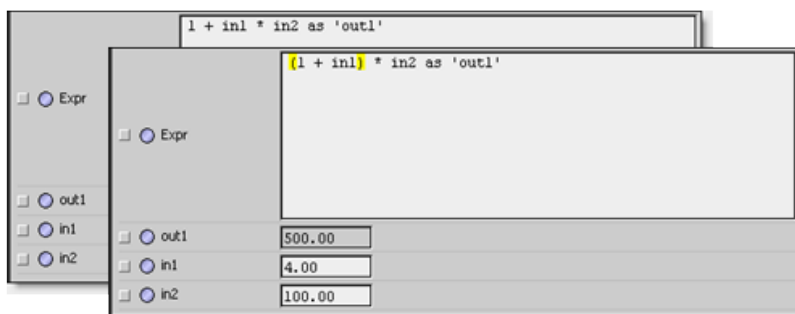
logical operators

6. as

result operator

You may use parentheses to override the normal precedence as illustrated in the following examples.

Figure 1. A change in precedence



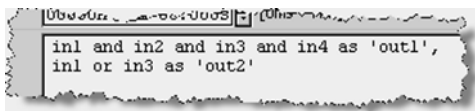
In the first expression, multiplication precedes addition. Adding the parentheses changes the precedence so that addition precedes multiplication.

Parent topic: [Syntax](#)

Commas

If creating a Expr component with multiple expressions (output slots), use a comma to delimit expression statements from one another.

Figure 1. Example of commas



Parent topic: [Syntax](#)

Long statements

When entering an expression, keep typing or enter a CR/LF to wrap a long statement.

Figure 1. Example of a CR/LF

```

in1 and in2 and in3 and in4 and in5 and in6
and in7 and in8 and in9 and in10 and in11 and
in12 and in13 and in14 and in15 and in16 as 'out1',
in1 or in3 as 'out2'

```

The example above consists of two expression statements: the first requires three lines; the second requires only a few characters.

Note: Outputs require surrounding apostrophes; inputs do not.

Parent topic: [Syntax](#)

Create a BQL Expression component

These topics describe different BQL Expression components and include some examples.

- [To create a BQL Expression component](#)
- [Mathematical expressions](#)
At the heart of Expr is a BQL expression that processes inputs and outputs. You have, for example, a component with three properties: inA, inB and outAdd. To add the two inputs together you would use this expression:
- [Logical expressions](#)
In the following example, two Boolean properties use an AND gate to output a Boolean value.
- [Component instances](#)
The kitControl palette contains multiple instances of the BQL Expression component with different default configurations.
- [More examples](#)

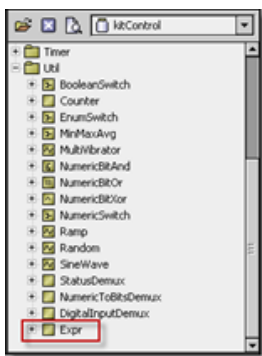
Parent topic: [BQL Expression component](#)

To create a BQL Expression component

1. Design your expression. This step answers the question, “What does Expr need to do?”

For example, say you wish to create a logic component with four Boolean inputs and two Boolean outputs. One output is an AND function on all inputs, the other output is an OR function on just two of the inputs.

2. Drag an Expr from the kitControl palette onto your wire sheet and give it a name.
Figure 1. kitControl components with BQL Expression (Expr) highlighted



3. Using the slot sheet view of the component, add four Boolean inputs and two Boolean outputs to the Expr and give them appropriate and unique names.

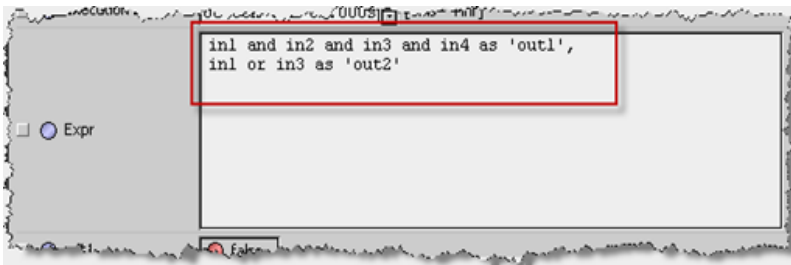
Note: When adding input slots, set (check) the **Execute On Change** flag for each one.
Figure 2. New slots (out1, out2, in1, in2, in3, in4)

| | | | | |
|-------------|-------|-------|---------|---------------------|
| Property 12 | Link3 | Link3 | Dynamic | baja:ConversionLink |
| Property 13 | out1 | out1 | Dynamic | rs baja:Boolean |
| Property 14 | out2 | out2 | Dynamic | rs baja:Boolean |
| Property 15 | in1 | in1 | Dynamic | sxl baja:Boolean |
| Property 16 | in2 | in2 | Dynamic | sxl baja:Boolean |
| Property 17 | in3 | in3 | Dynamic | sxl baja:Boolean |
| Property 18 | in4 | in4 | Dynamic | sxl baja:Boolean |

- Using the property sheet view, configure any execution delay.

The expression will run after this delay, which is useful to throttle rapidly changing properties used as inputs.

- Continuing on the property sheet, enter the BQL statements in the Expr field.
Figure 3. Example expression statements



All other properties are assumed to be dynamic properties and can be added or removed on the Expr slot sheet.

- Click **Save**, then the “Up Level” menu bar icon to return to the parent wire sheet. The new Expr component is visible, but without any “pinned” slots for inputs and outputs.

Although optional, you can pin slots before linking, by right-clicking the component and selecting **Pin Slots**. This can speed up linking, as otherwise the popup **Link** dialog appears if linking to unpinned slots. You can also use the right-click **Reorder** option to change the top-to-bottom positioning of slots.

- Using the wire sheet view, make links to the slots and test your logic.
Figure 4. Example of an expected result

| ANDOR Bql Expr Component | |
|--------------------------|-------|
| out1 | false |
| out2 | true |
| in1 | true |
| in2 | true |
| in3 | false |
| in4 | true |

The figure above shows all slots of this example Expr component linked.

Parent topic: [Create a BQL Expression component](#)

Mathematical expressions

At the heart of Expr is a BQL expression that processes inputs and outputs. You have, for example, a component with three properties: inA, inB and outAdd. To add the two inputs together you would use this expression:

inA + inB as 'outAdd'

In mathematical terms, this means:

inA + inB = outAdd

In the example, inA and inB must use the executeOnChange slot flags. When inA or inB change, Expr executes and updates outAdd.

Expr is not restricted to a limited number of properties. You can add and remove properties via Expr's slot sheet. For example, using the above, we could add another two properties called outMult and inC. The expression would now read:

inA + inB as 'outAdd',

inA * inB * inC as 'outMult'

The comma at the end of the first expression indicates that another expression follows. Use the comma to create multiple expressions that update multiple outputs.

Parent topic: [Create a BQL Expression component](#)

Logical expressions

In the following example, two Boolean properties use an AND gate to output a Boolean value.

inBoolA and inBoolB is 'out'

Again, the inBoolA and inBoolB must use the executeOnChange slot flag.

Note: Linking slots with different data types automatically inserts a conversion link between the two. For example, you may link a StatusNumeric property to a Double property. For more information, see *Conversion Links*.

Parent topic: [Create a BQL Expression component](#)

Component instances

The kitControl palette contains multiple instances of the BQL Expression component with different default configurations.

- Util folder

A blank Expr is its default state.

- Logic folder

Provides Expr with four Boolean inputs and one Boolean output. By default, all inputs are joined using AND.

- Math folder

Provides Expr with four Double inputs and one Double output. By default, all inputs are mathematically added together.

Note: Any of these components can be modified by adding and removing properties using the slot sheet.

Parent topic: [Create a BQL Expression component](#)

More examples

- Two inputs, logical AND

The screenshot shows the configuration window for the 'AND 2 (Bql Expr Component)'. The 'Status' is set to '(ok)'. The 'Fault Cause' is empty. The 'Execution Delay' is set to '00000h 00m 00.000s' with a range of '[0ms - +inf]'. The 'Expr' field contains the text 'inA and inB as 'out''. The 'inA' input is set to 'true' (indicated by a green circle), 'inB' is set to 'false' (indicated by a red circle), and the 'out' is set to 'false' (indicated by a red circle).

- Not two inputs

The screenshot shows the configuration window for the 'NAND 2 (Bql Expr Component)'. The 'Status' is set to '(ok)'. The 'Fault Cause' is empty. The 'Execution Delay' is set to '00000h 00m 00.000s' with a range of '[0ms - +inf]'. The 'Expr' field contains the text 'not (inA and inB) as 'out''. The 'inA' input is set to 'true' (indicated by a green circle), 'inB' is set to 'false' (indicated by a red circle), and the 'out' is set to 'true' (indicated by a green circle).

- Two-input addition

ADD (Bq Expr Component)

☐ Status (ok)

☐ Fault Cause

☐ Execution Delay 00000h 00m 00.000s [0ms - +inf]

☐ Expr

☐ out1 123.00

☐ in1 23.00

☐ in2 100.00

in1 + in2 as 'out1'

- Divider (two Double) properties

DIVIDE (Bq Expr Component)

☐ Status (ok)

☐ Fault Cause

☐ Execution Delay 00000h 00m 00.000s [0ms - +inf]

☐ Expr

☐ out1 6.67

☐ in1 100.00

☐ in2 15.00

in1 / in2 as 'out1'

- Subtraction

SUBTRACT (Bq Expr Component)

☐ Status (ok)

☐ Fault Cause

☐ Execution Delay 00000h 00m 00.000s [0ms - +inf]

☐ Expr

☐ out1 -100.00

☐ in1 0.00

☐ in2 100.00

in1 - in2 as 'out1'

- Expression mixture

MIX 1 (Bq Expr Component)

☐ Status: {ok}

☐ Fault Cause:

☐ Execution Delay: 00000h 00m 00.000s [0ms - +inf]

☐ Expr:

inA and inB as 'out1',
inX + inY as 'out2'

☐ inX: 10.00

☐ inY: 20.00

☐ out2: 30.00

☐ inA: true

☐ inB: false

☐ out1: false

- Greater and less than expressions

GREATER OR LESS THAN (Bq Expr Component)

☐ Status: {ok}

☐ Fault Cause:

☐ Execution Delay: 00000h 00m 00.000s [0ms - +inf]

☐ Expr:

in1 < 10 as 'out1',
in1 > 10 as 'out2'

☐ out1: 0.00

☐ in1: 10.52

☐ in2: 0.00

☐ out2: true

GREATER OR LESS THAN (Bq Expr Component)

☐ Status: {ok}

☐ Fault Cause:

☐ Execution Delay: 00000h 00m 00.000s [0ms - +inf]

☐ Expr:

in1 < 10 as 'out1',
in1 > 10 as 'out2'

☐ out1: 1.00

☐ in1: 9.16

☐ in2: 0.00

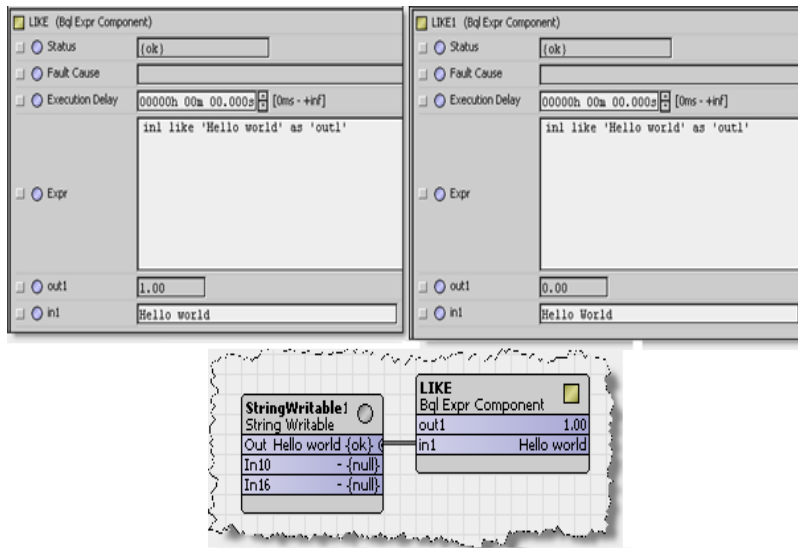
☐ out2: false

| Property | Link | Link | Dynamic | baja:ConversionLink |
|-------------|------|------|---------|---------------------|
| Property 9 | in1 | in1 | Dynamic | sxL |
| Property 10 | in2 | in2 | Dynamic | sx |
| Property 11 | out1 | out1 | Dynamic | rs |
| Property 12 | out2 | out2 | Dynamic | rs |

The examples above each result in two outputs.

The last column on the right indicates that the two outputs are different slot types (Double and Boolean). This is legal.

- Using the “like” expression



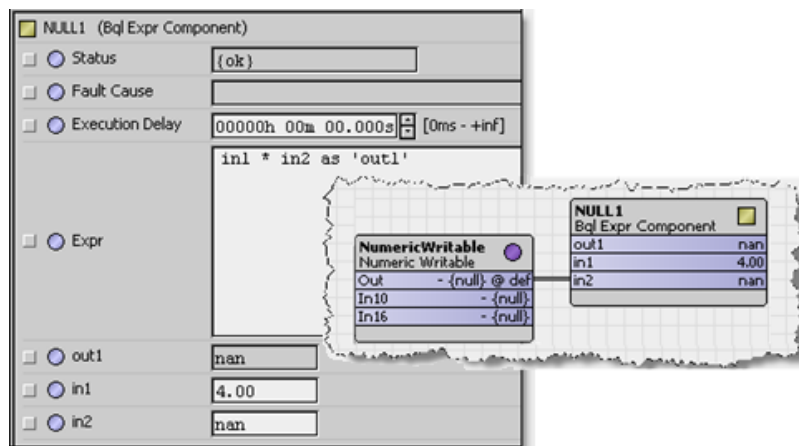
- Multiply four Double properties
inA * inB * inC * inD as 'out'
- Multiply two BStatusNumeric properties to a Double output property.
inA.value * inB.value as 'out'
- Negate a Double input property (if inA is 5, out becomes -5)
-inA as 'out'

Parent topic: [Create a BQL Expression component](#)

Handling null

This section describes expression handling of a “null” input from a linked status type output.

Figure 1. The result of a math expression with any null (or nan) input can be nan



In a math expression, if a Double or Float input slot is linked to a source StatusNumeric output with a “null”

value, the input is evaluated as “nan” (not a number). If the expression has a Double or Float slot as output, the result is also nan, as shown. This also occurs if such an input has a nan.

Note if the input slot is an Integer or Long type, the expression ignores the null value—the last valid value is used (or if nan input, is processed as value 0). The output is some number. If an input slot is a StatusNumeric (requires expression syntax `inputSlotName.value`), a null input is seen but not processed.

In a logic expression, if a Boolean input slot is linked to a source BooleanNumeric output with a “null” value, the null is ignored by the expression—the last valid value is used. If the input slot is a StatusBoolean type (again, syntax `inputSlotName.value` is required), the null is seen but not processed.

The Expr component utilizes the automatic “conversion links” feature introduced in AX-3.6, such that “link from” behavior between dissimilar data types is followed. For more information, see *Conversion Links*.

In general, if creating an Expr component for use in control logic that may have one or more “null” inputs, it is recommended that you test it to verify the desired behavior.

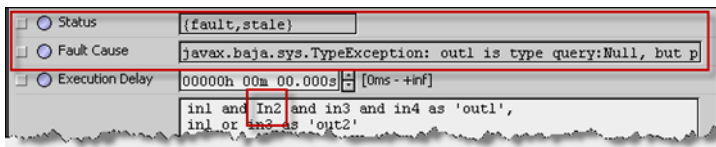
Note: Starting in AX-3.6, changes were also made to the various “status value to simple value” kitControl Conversion components, to allow the option of specifying an output value when the status input is null. If needed, you may wish to use one or more of these components “upstream” of the Expr component. For more details, refer to the *NiagaraAX KitControl Guide* section “Status value to simple value”.

Parent topic: [BQL Expression component](#)

Troubleshooting

If you enter the wrong syntax, or Expr recognizes that you are trying something illegal, it displays status and fault cause information.

Figure 1. Example of a fault



The information fields have the following meanings:

- **Status**
Reports problems with the expression. The expression may be invalid or one of the properties it references may not exist.
- **Fault Cause**
Gives a verbal description of expression errors.

After correcting the fault, click **Save** to restart the Expr.

Note: No compilation is required.

Parent topic: [BQL Expression component](#)

Frequently-asked questions

How many inputs and outputs does Expr allow?

Expr supports as many inputs and outputs as you need. Two specific instances of Expr, located in the Logic and Math folders, each support four inputs and a single output.

| Frequently Asked Question | Answers |
|---|---|
| Does an Expr need to be compiled before it can be used on a wire sheet? | No, if you make a change, click Save. No compilation is required. |
| Can I create more than one expression per component? | Yes. You may create as many expressions as you need, delimited by commas. |
| Can I put comments into an Expr? | No. Comments are not allowed in a BQL Expression component. |
| Can I use stored variables in an Expr? | No. To store variables, use a Program component. |
| Can Expr handle time functions? | No. Time functions require a Program component. |
| What types of operations does the Expr support? | The Expr supports math and logic operators. |

Parent topic: [BQL Expression component](#)

Conversion Links

Starting in AX-3.6, linking components no longer requires that slots have matching data types, nor usage of special components found in the “Conversion” folder in the kitControl palette. Now in most cases, you simply link between a source slot and target slot, regardless of data types, and a conversion link is automatically created to handle conversion.

Each conversion link has a “BIConverter” implementation as a child, which is used to manage and customize the conversion process.

Conversion links can simplify control logic by reducing amounts of needed components. Conversion links can also help to de-clutter wire sheets.

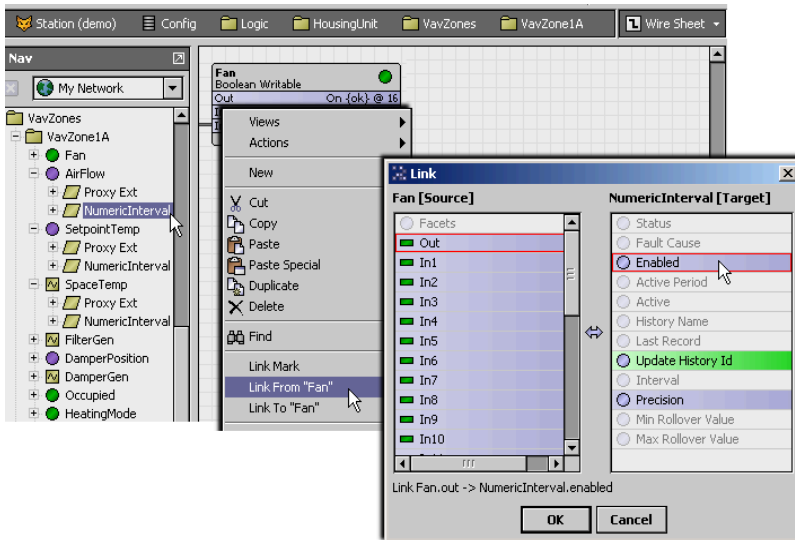
Note: Conversion links require AX-3.6 or higher on “both ends” when working with engineering control logic. That is, the host (JACE) running the station must be at AX-3.6 or higher, and Workbench must be at AX-3.6 or higher. Otherwise, conversion links are unavailable.

- [Typical conversion link usage](#)
The most expected usage of a conversion links is to support a link between a “status type” numeric or boolean to a “simple type” numeric or boolean, or vice versa.
- [Data transformation via conversion link](#)
Many types of conversion links go beyond the “simple-to-status type” or “status type-to-simple” model used in kitControl “Conversion” components. The figure below shows one example.
- [Supported conversion link types](#)
In AX-3.6 and later, a conversion link automatically results if you link two slots with dissimilar data types. Table 1 lists those conversion-supported (Y) links, by “From” and “To” slots, by data type.
- [Converter components](#)
Any conversion link has a child “Converter” component. To see it in Workbench, open the **Edit** dialog for the link.
- [Converter properties](#)
Most conversion links have no properties under the link’s Converter component. This is true for all conversion links between “like” data types, that is, status type-to simple and vice versa. However, conversion links that transform values like string to boolean or numerical, or the reverse (to string) may have one or more Converter properties.
- [Conversion link "From" notes](#)
These notes describe some conversion link behaviors, grouped by source (From) data types:

Typical conversion link usage

The most expected usage of a conversion links is to support a link between a “status type” numeric or boolean to a “simple type” numeric or boolean, or vice versa.

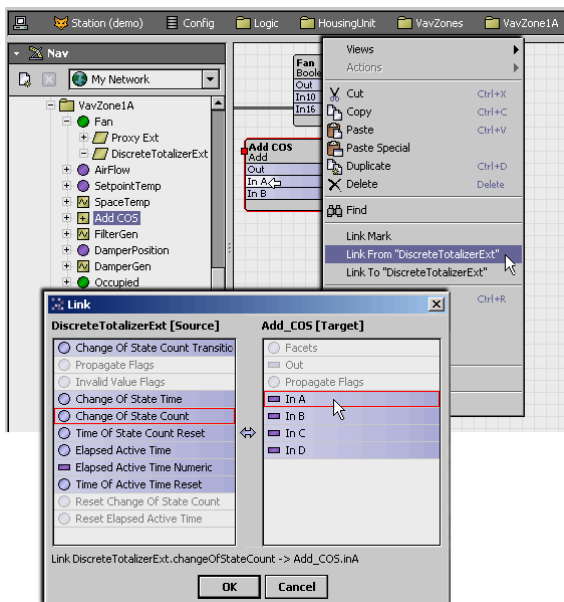
Figure 1. Example link from status type to simple type (statusBoolean to Boolean)



In the example above, the statusBoolean “out” of a BooleanWritable (Fan control) point is needed to enable/disable several history extensions, where the “Enabled” property is a simple boolean type. Before AX-3.6, this required an intervening “StatusBooleanToBoolean” component. However, now you simply link between the two dissimilar slots, as allowed in the **Link** editor.

The example below shows linking from a simple data type to a status type.

Figure 2. Example link from simple type to status type (integer to statusNumeric)



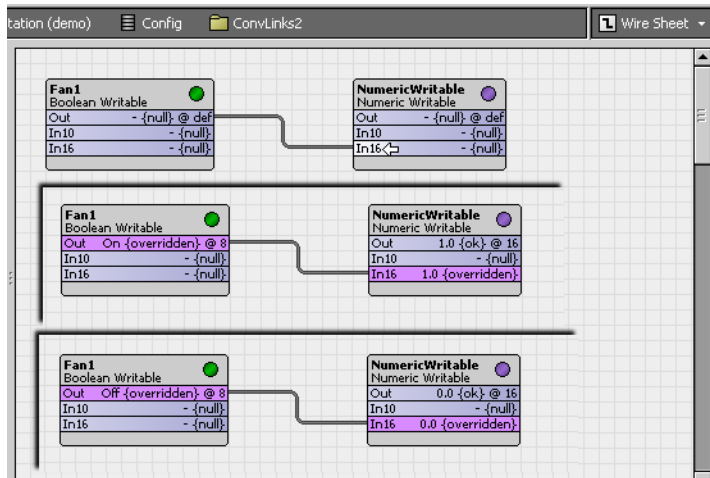
In the example above, the integer “Change Of State Count” slot value of a DiscreteTotalizerExt of a BooleanWritable is needed as an input in a Math component, say an “Add” component. Before AX-3.6, this required an intervening “IntegerToStatusNumeric” component. Again, now you can simply link the two slots, as allowed in the **Link** editor.

Parent topic: [Conversion Links](#)

Data transformation via conversion link

Many types of conversion links go beyond the “simple-to-status type” or “status type-to-simple” model used in kitControl “Conversion” components. The figure below shows one example.

Figure 1. Example link from statusBoolean to statusNumeric

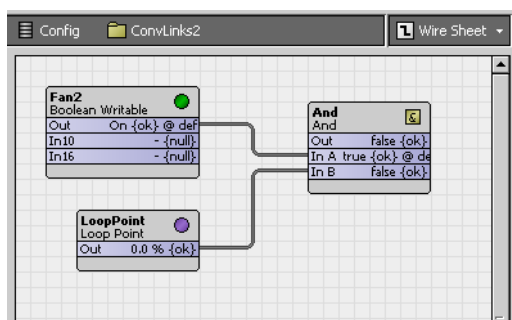


In the example above, the statusBoolean “out” of a BooleanWritable is linked to the statusNumeric “In16” slot of another component. With no other configuration, this transforms the boolean “active” value to a numeric “1”, a boolean “inactive” value to a numeric “0”, and passes through a “null” value.

This sort of “value transform” may be useful in downstream logic. Note before AX-3.6, this type of conversion was often done via a custom Program component.

To reverse this example, you can link the statusNumeric “out” of a component to a statusBoolean slot of another component, for example a kitControl “Logic” component. See the figure below.

Figure 2. Example link from statusNumeric to statusBoolean



In this case, the background “Status Numeric To Status Boolean” converter works like this:

- Numeric value of “0” is boolean “false”.
- Numeric value not “0” is boolean “true” (note this means value > 0, and also a negative value, i.e. < 0).

Many other link converters are used in various link combinations, including many string-related ones and time-related ones.

For more details, see the following topics:

- “Supported conversion link types” for a matrix of allowed links by data types.
- “Converter components” for details on the converter that is automatically selected.
- “Converter properties” for information on possible converter properties.
- “Conversion link "From" notes” for details on linking from some specific data types.

Parent topic: [Conversion Links](#)

Supported conversion link types

In AX-3.6 and later, a conversion link automatically results if you link two slots with dissimilar data types. Table 1 lists those conversion-supported (Y) links, by “From” and “To” slots, by data type.

Table 1. Supported conversion links, by “From” data type and “To” data type

| | | From | | | | | | | | | | | |
|----|---------------|--------|---------|--------|-------|------|---------|------------|-------------|---------------|---------------|------------|--------------|
| | | string | boolean | double | float | long | integer | frozenEnum | dynamicEnum | statusBoolean | statusNumeric | statusEnum | statusString |
| To | string | — | Y* | Y* | Y* | Y* | Y* | Y | Y* | Y | Y | Y | Y |
| | boolean | Y* | — | Y* | Y* | Y* | Y* | Y | Y* | Y | Y* | Y* | Y |
| | double | Y | Y | — | Y | Y | Y | Y | Y | Y* | Y | Y | Y |
| | float | Y | Y | Y | — | Y | Y | Y | Y | Y* | Y | Y | Y |
| | long | Y | Y | Y | Y | — | Y | Y | Y | Y* | Y | Y | Y |
| | integer | Y | Y | Y | Y | Y | — | Y | Y | Y* | Y | Y | Y |
| | frozenEnum | — | — | — | — | — | — | — | — | — | — | — | — |
| | dynamicEnum | — | Y | Y | Y | Y | Y | Y | — | Y | Y | Y | — |
| | statusBoolean | Y* | Y* | Y* | Y* | Y* | Y* | Y | Y* | — | Y | Y* | Y |
| | statusNumeric | Y | Y | Y | Y | Y | Y | Y | Y | Y | — | Y | Y |
| | statusEnum | — | Y | Y | Y | Y | Y | Y | Y | Y | Y | — | — |
| | statusString | Y | Y | Y* | Y* | Y* | Y* | Y | Y | Y | Y* | Y | — |
| | ord | Y | — | — | — | — | — | — | — | — | — | — | Y |
| | time | — | — | Y | Y | Y | Y | — | — | — | — | — | — |
| | absTime | Y | — | Y | Y | Y | Y | — | — | — | Y | — | Y |
| | relTime | — | — | Y | Y | Y | Y | — | — | — | Y | — | — |

Each of the “From-To” links made above results in a conversion link, where that link (component) has a specific type of child Converter component.

Note: Y* indicates that the Converter component of the link has one or more properties.

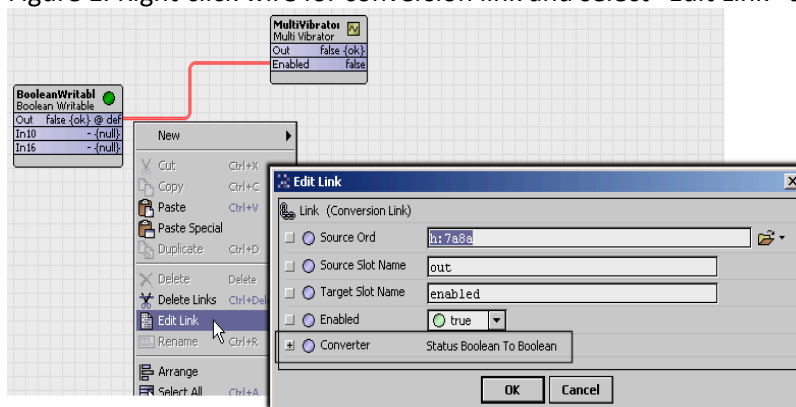
Parent topic: [Conversion Links](#)

Converter components

Any conversion link has a child “Converter” component. To see it in Workbench, open the **Edit** dialog for the link.

- If the link shows as a wire on the wire sheet, right-click it and select “**Edit Link**”.

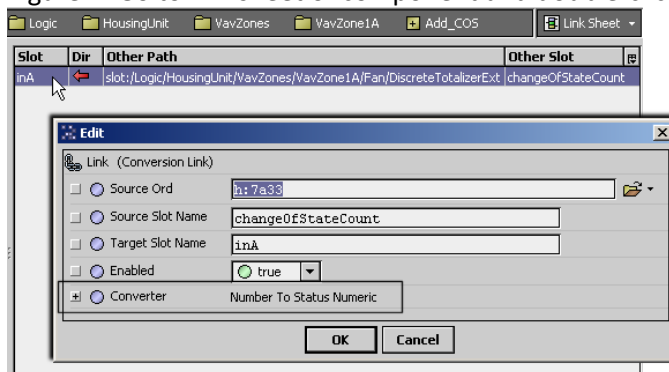
Figure 1. Right-click wire for conversion link and select “Edit Link” to see converter



The conversion link above uses converter type “Status Boolean To Boolean”.

- If the link shows on the wire sheet as a knob on a component, go to the component’s link sheet view, then double-click the desired link for its **Edit** dialog. See below.

Figure 2. Go to link sheet of component and double-click conversion link for “Edit” dialog



The link above is a “Number to Status Numeric” converter, allowing the integer “Change Of State Count” value from a DiscreteTotalizeExt to be used in a Math component (from the example shown in the topic “Conversion Link Usage”).

Typically, you seldom need to edit links between components. One exception is when linking from the dynamically-created components of a station’s PlatformServices to other components, where you need to edit the “Source Ord” property from “Handle” to “Slot”. For related details, refer to the *NiagaraAX Platform Guide* section “PlatformServices binding and link caveats”.

Another possible exception may be with some conversion links, where a child Converter component has one or more properties (such links are indicated with a “Y*” in the Table 1 matrix of supported links). See “Converter properties”.

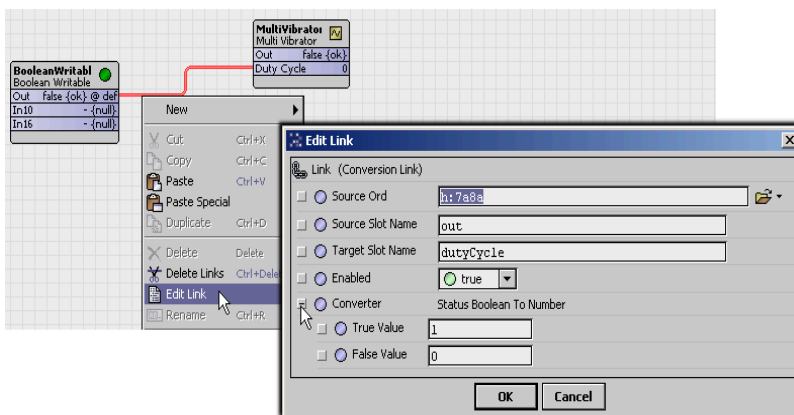
Parent topic: [Conversion Links](#)

Converter properties

Most conversion links have no properties under the link’s Converter component. This is true for all conversion links between “like” data types, that is, status type-to simple and vice versa. However, conversion links that transform values like string to boolean or numerical, or the reverse (to string) may have one or more Converter properties.

The image below shows one example.

Figure 1. Example Converter with properties (Status Boolean To Number)



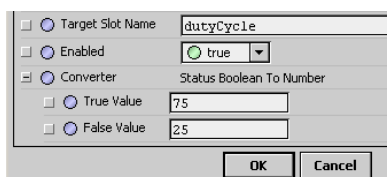
The example above reflects default values for a link made from a statusBoolean slot (in this case, “out” of a BooleanWritable) to an Integer slot (in this case, “Duty Cycle” of a MultiVibrator). The converter type automatically used is “Status Boolean To Number”.

Expanding the Converter component in the **Edit Link** dialog reveals two properties:

- True Value — default value of 1
- False Value — default value of 0

The duty cycle range of a MultiVibrator is from 0 to 100, so defaults here are not typically appropriate.

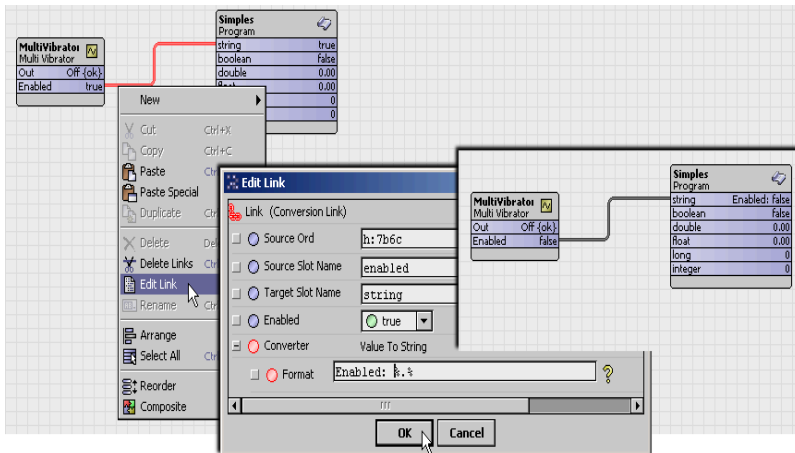
Figure 2. Example Converter properties edited to non-default values



In this example, converter properties were edited to true at 75, and false at 25, as shown in the figure above.

In cases where the link’s slot target (To) data type is string or statusString, the Converter property, if present, is typically “Format”. If a default “%. %” value, this means it is BFormat (Baja Format).

Figure 3. Example Format property of “Boolean To String” Converter edited from default “%.%”



The figure above shows the static text “Enabled:” prepended on the default % . % Format property value of a “Boolean to String” link Converter.

Parent topic: [Conversion Links](#)

Conversion link "From" notes

These notes describe some conversion link behaviors, grouped by source (From) data types:

See the following topics for more details.

- [Links from string](#)
- [Links from boolean](#)
- [Links from double, float, long, integer](#)
The following summarize links from one of the simple number data types (double, float, long, or integer, described below as “number”), to another data type:
- [Links from statusBoolean](#)
- [Links from statusNumeric](#)
- [Links from statusEnum](#)
- [Links from statusString](#)
- [Links from time](#)
- [Links from absTime](#)
- [Links from relTime](#)

Parent topic: [Conversion Links](#)

Links from string

- A string-to-boolean link or string-to-statusBoolean link has a “False Value” converter property, with a default value of `false`. Any other source string value, instead of (or in addition to) this “False Value” string (case insensitive) results in the target boolean or statusBoolean value to be true. Note that a blank string source value is also a boolean or statusBoolean true (unless the “False Value” is blank).
- A string-to-double or string-to-float link requires a source string using decimal numerals only, expressed as a decimal number or integer, either positive or negative. Any other source string characters (or blank string) results in the linked double or float slot to have an “nan” (not a number) value.
- A string-to-long or string-to-integer link requires a source string expressed as an integer only, either

positive or negative. Any additional characters in the source string results in the linked long or integer slot to have a 0 value, or the last non-zero value.

- A string-to-statusNumeric link requires a source string using decimal numerals only, expressed as a decimal number or integer, either positive or negative. Any other source string characters (or blank string) results in the linked statusNumeric slot to have a “fault” condition, with no value change.
- A string-to-statusString link results in that string value in the statusString.
- A string-to-ord link provides no Niagara ord format/error checking—use sparingly.
- A string-to-absTime link requires an ISO-formatted string—otherwise, the linked absTime value remains null. The ISO format is “yyyy-mm-ddThh:mm:ss.mmm[+/-]hh:mm”, for example, “2011-01-31T13:23:53.772-05:00”, which an absTime slot could show as “31-Jan-2011 01:23 PM EST”.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link “From” notes](#)

Links from boolean

- A boolean-to-string results in either a “false” string value if boolean false, or “true” string value if boolean true. The link has a “Format” converter property. See the “Converter Properties topic for an example.
- A boolean-to-simple number data type link (to-double, to-float, to-long, to-integer) results in a 0 value for a boolean false, or 1 if a boolean true.
- A boolean-to-statusBoolean link has a “False Value” converter property, with a default value of 0. The default 0 keeps the statusBoolean value in synch with the source boolean value. If “False Value” is set to 1, the linked statusBoolean value is opposite (NOT) the source.
- A boolean-to-statusNumeric link or boolean-to-statusEnum link results in a 0 value for a boolean false, or 1 if a boolean true.
- A boolean-to-statusString has a string value `false` if false, or string value `true` if boolean true.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link “From” notes](#)

Links from double, float, long, integer

The following summarize links from one of the simple number data types (double, float, long, or integer, described below as “number”), to another data type:

- A number-to-string or number-to-statusString link results in the string value of that number, for example “78.30” The link also has a “Format” converter property based on a text string, with a blank default value. The blank Format outputs all existing digits with no formatting.

In the Format property, you can enter a Format value using pound signs (#) for digits, 0 numeral(s) for leading/trailing zero(s), and placeholder separators, such as a comma (,) for grouping and/or period (.) as decimal separator. Some Format value examples:

– ###,###.### — where a source number 123456.789 is string formatted as 123,456.789

- ###, ## — where a source number 123456.789 is string formatted as 123456.79
- 00000.000 — where a source number 123.78 is string formatted as 000123.780
- A number-to-boolean link or number-to-statusBoolean link has a “False Value” converter property, with a default value of 0, such that any number value other than 0 results in a boolean true. If needed, “False Value” can be edited to specify a different value to associate with false.
- A number-to- “different simple number type” link results in that number, unless outside the range of the target data type. For example, a double-to-integer link with a source value of 2147484000 (exceeding max integer value of 2147483647) will result in the integer value of “max” (2147483647).
- A number-to-statusNumeric link results in that number.
- A number-to-statusEnum link results in that number, unless outside the ordinal (integer) range of a statusEnum data type. For example, a double-to-statusEnum link with a source value of 2147484000 (exceeding max integer value of 2147483647) will result in an Enum (ordinal) value of 2147483647.
- A number-to-Time link results in that number of milliseconds added to the base time of 12:00am midnight (00:00). For example, a float value of 900000 is seen as Time 12:15am (00:15).
- A number-to-absTime link adds that number of milliseconds to the Java “epoch” date/timestamp of December 31 1969 7pm EST. For example, a long value of 1296509138929 results in an absTime value of January 31 2011 4:25pm EST.
- A number-to-relTime link adds that number of milliseconds to 0ms. For example, an integer value of 4800000 results in an relTime value of 1hour 20mins, or -108000 results in -1min 48sec.

Links to other “Conversion link "From" notes”.

Parent topic: [Conversion link "From" notes](#)

Links from statusBoolean

- A statusBoolean-to-string results in either a “false” string value if boolean false, or “true” string value if boolean true. A statusBoolean “null” value does not change the target string value.
- A statusBoolean-to-simple number data type link (to-double, to-float, to-long, to-integer), by default, results in a 0 value for a boolean false, or 1 if a boolean true. A link’s Converter has two editable child properties: “True Value” (default = 1) and “False Value” (default = 0), to specify non-default values.

Typically, a statusBoolean “null” does not change a target long or integer link 0 value; however, a “null” changes a linked double or float value to “nan” (not a number).
- A statusBoolean to statusNumeric or statusEnum link results in a 0 value for a boolean false, or 1 if a boolean true. A statusBoolean “null” changes the statusNumeric or statusEnum target to “null”.
- A statusBoolean-to-statusString, by default, has a string value `false` if false, or string value `true` if boolean true. A statusBoolean “null” changes the statusString target to “null”.

Links to other “Conversion link "From" notes”.

Parent topic: [Conversion link "From" notes](#)

Links from statusNumeric

- A statusNumeric-to-string link results in the string value of that number, for example “78.30”. A statusNumeric “null” value does not change the target string value.
- A statusNumeric-to-boolean link has a “False Value” converter property, with a default value of 0 . 0, such that any number value other than 0 results in a boolean true. If needed, “False Value” can be edited to specify a different value to associate with false.

Typically, a statusNumeric “null” value does not change the target boolean value.

- A statusNumeric to “different simple number type” link results in that number, unless outside the range of the target data type. For example, a statusNumeric-to-integer link with a source value of 2147484000 (exceeding max integer range) will result in the integer value of “max” (2147483647).

A statusNumeric “null” value does not change/affect a linked long or integer value; however, a “null” changes a linked double or float value to “nan” (not a number).

- A statusNumeric-to-statusBoolean link has a “False Value” converter property, with a default value of 0 . 0, such that any number value other than 0 results in a boolean true. If needed, “False Value” can be edited to specify a different value to associate with false.

A statusNumeric “null” value changes the statusBoolean target to “null”.

- A statusNumeric-to-statusEnum link results in that number, unless outside the ordinal (integer) range of a statusEnum data type, whereby it is “clamped” at that max or min value.

A statusNumeric “null” value changes the statusEnum target to the “null value” (often 0).

- A statusNumeric-to-statusString link results in the string value of that number, for example “78.30”. The link also has a “Format” converter property, based on a text string, with a blank default value. The blank Format outputs all existing digits with no formatting.

In the Format property, you can enter a Format value using pound signs (#) for digits, 0 numeral(s) for leading/trailing zero(s), and placeholder separators, such as a comma (,) for grouping and/or period (.) as decimal separator. Some Format value examples:

- ###,###.### — where a source number 123456.789 is string formatted as 123,456.789
- ###,## — where a source number 123456.789 is string formatted as 123456.79
- 00000.000 — where a source number 123.78 is string formatted as 000123.780

A statusNumeric “null” value changes the linked statusEnum target to “null”.

- A statusNumeric-to-absTime link adds that number of milliseconds to the Java “epoch” date/timestamp of December 31 1969 for another date-timestamp. For example, a statusNumeric value of 1296509138929 results in an absTime value of January 31 2011 4:25pm EST.

A statusNumeric “null” value changes the absTime value to “null”.

- A statusNumeric-to-relTime link adds that number of milliseconds to 0ms. For example, a statusNumeric value of 4800000 results in an relTime value of 01h 20m.

A statusNumeric “null” value does not change the current linked relTime value.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link "From" notes](#)

Links from statusEnum

- A statusEnum-to-string link results in the string value of the Enum's tag (descriptor), for example "Off" or "Occupied". A statusEnum "null" value does not change the target string value.
- A statusEnum-to-boolean link has a "False Value" converter property, with a default value of 0, such that any ordinal value other than 0 results in a boolean true. If needed, "False Value" can be edited to specify a different ordinal (integer) Enum value to associate with false.

A statusEnum "null" value does not change the target boolean value.

- A statusEnum to "different simple number type" link results in the (integer) ordinal value of that Enum. A statusEnum "null" value does change/affect a linked long or integer value; however, a "null" changes a linked double or float value to "nan" (not a number).
- A statusEnum-to-statusBoolean link has a "False Value" converter property, with a default 0 value, such that any ordinal value other than 0 results in a boolean true. If needed, "False Value" can be edited to specify a different ordinal (integer) Enum value to associate with false.

A statusEnum value "null" value changes the statusBoolean target to "null".

- A statusEnum to statusNumeric link results in the (integer) ordinal value of that Enum. A statusEnum "null" value changes the statusEnum target to "null".
- A statusEnum-to-statusString link results in the string value of the Enum's tag (descriptor), for example "Off" or "Occupied". A statusEnum "null" value changes the statusString target to "null".

Links to other "Conversion link "From" notes".

Parent topic: [Conversion link "From" notes](#)

Links from statusString

- A statusString-to-string link results in that string value in the statusString.
- A statusString-to-boolean link has a "False Value" converter property, with a default value of `false`. Any other source string value, instead of (or in addition to) this "False Value" string (case insensitive) results in the target boolean value to be true. Note that a blank statusString source value is also a boolean true (unless the "False Value" has been set to blank).
- A statusString-to-double or statusString-to-float link requires a source string using decimal numerals only, expressed as a decimal number or integer, either positive or negative. Any other source string characters (or blank string) results in the linked double or float to have a "nan" (not a number) value.
- A statusString-to-long or statusString-to-integer link requires a source string expressed as an integer only, either positive or negative. Any additional characters in the source string results in the linked long or integer slot to have a 0 value, or the last non-zero value.
- A statusString-to-statusNumeric link requires a source string using decimal numerals only, expressed as a decimal number or integer, either positive or negative. Any other source string characters (or blank string) results in the linked statusNumeric to have a "fault" condition, and no value change.
- A statusString-to-ord link provides no ord format/error checking—use sparingly.

- A `statusString-to-absTime` link requires an ISO-formatted string—otherwise, the linked `absTime` value remains null. The ISO format is “`yyyy-mm-ddThh:mm:ss.mmm[+/-]hh:mm`”, for example, “`2011-01-31T13:23:53.772-05:00`”, which an `absTime` slot could show as “`31-Jan-2011 01:23 PM EST`”.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link “From” notes](#)

Links from time

- A `time-to-string` link results in a string value that (by default) reflects `hr:min:sec.ms`, for example a time of `3:31 PM` could result in a string value of `15:31:23.647`. The link has “Format” converter property, with a default value of `HH:mm:ssZ`. You can edit this if needed—for example reducing Format to `HH:mm` for string output like `15:31`, or to `HH:mm a` for string output `3:31 PM`.
- A `time-to-“simple number type”` link (double, float, long, integer) results in the number of milliseconds in the current time since the base time of 12:00 AM midnight (00:00). For example, a `time-to-integer` link from `11:30 AM` would have a value of `41400000`.
- A `time-to-absTime` link results in the current date and time in the `absTime` value.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link “From” notes](#)

Links from absTime

- An `absTime-to-string` or `absTime-to-statusString` link results in a string value for a date-timestamp that (by default) reflects `yyyy-mo-dayThr:min:sec.ms-tzone offset hr`, for example a value of `2011-02-01T13:47:13.358-05:00`. The link has “Format” converter property, with a default value of `YYYY-MM-DDTHH:mm:ssZ`. You can edit this if needed—for example to change Format to `MM-DD-YYYY HH:mm a` to get a string output like `02-01-2011 13:51 PM`.
- An `absTime-to-“simple number type”` link (double, float, long, integer) results in the number of milliseconds for that date-timestamp since the Java “epoch” timestamp of December 31 1969.
- An `absTime-to-statusNumeric` link results in the number of milliseconds for that date-timestamp since the Java “epoch” timestamp of December 31 1969.
- An `absTime-to-time` link results in the time portion of the date-timestamp to be the time value.

Links to other “Conversion link “From” notes”.

Parent topic: [Conversion link “From” notes](#)

Links from relTime

- A `relTime-to-string` link or `relTime-to-statusString` link results in a string describing the relative time. For example, a `relTime` of `25h 20m 02s` can result in a string value of `1day 1hour 20mins 2sec`.
- A `relTime-to-“simple number type”` link (double, float, long, integer) reflects the number of milliseconds in the relative time. For example, a `relTime` value of `01h 20m` results in a value of `4800000`.
- A `relTime-to-statusNumeric` link reflects the number of milliseconds in the relative time. For example, a

relTime value of 01h 20m results in a value of 4800000.

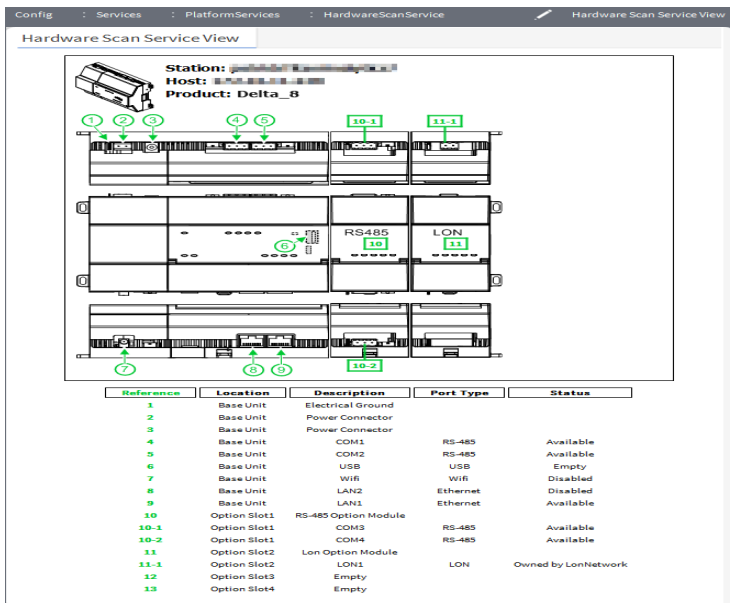
Links to other "Conversion link "From" notes".

Parent topic: [Conversion link "From" notes](#)

JACE Hardware Scan Service

The Hardware Scan Service is supported by NiagaraAX and Niagara 4. This optional service must be added under a JACE station's PlatformServices. The default view for the service shows a diagram of the hosting controller, identifying the location of communication ports and other features.

Figure 1. Hardware Scan Service View for the JACE-8000 with attached option modules



The diagram includes numbered callouts. The table below the diagram describes each callout (such as COM2, USB, etc.), port type, and status.

Also described are the types of option cards or option modules installed in option slots of the controller (if applicable), plus other information on features of the controller, such as the current “Wifi” status.

- [Hardware scan benefits](#)
- [Currently supported platforms](#)
Currently supported platforms are listed below, showing a thumbnail of the “controller image” portion of the service’s default **Hardware Scan Service View**.
- [Adding the HardwareScanService](#)
You can add the Hardware Scan Service in any JACE controller running AX-3.7 or later (including Niagara 4). To add the service, simply install the necessary software modules, using the **Software Manager** in a platform connection:
- [Hardware Scan Service View notes](#)
The main usage of the service is expected from its default view, the **Hardware Scan Service View**. Regardless of the JACE controller type, this view consistently appears with one or more image views of the physical controller, with callouts described in a table below.
- [HardwareScanService properties](#)
The HardwareScanService has a number of properties, available by selecting its property sheet from the view selector, as shown. Values from many of these properties are reflected in the graphical default view.
- [Lexicon customizing of HardwareScanService](#)
You can customize text descriptors and values that appear in the service’s **Hardware Scan Service View** by editing the lexicon for the platHwScan module and various platHwScanType modules.
- [Px customization](#)
You can make a Px view in a station that provides a customized alternative to the default **Hardware Scan Service View**, by using Px widgets available in the palette of the platHwScan module, as well

any custom edited image (.png) file or files (for the controller “layout” diagrams).

Hardware scan benefits

Note: The HardwareScanService is unlicensed, requiring only a JACE controller running AX-3.7 or later, with the appropriate modules installed.

Prior to this service, the hardware configuration of the JACE host was often known to its station (and Workbench) in a generic way, for example by its “Host Model” value of “NPM2” or “NPM6”. Such values refer to a “Niagara Processor Module” board that fits on different types of controller base boards. Thus, an “NPM2” host could be either JACE-2, JACE-202 Express (M2M JACE), or a Security JACE.

Although a station’s platform SerialPortService lists each serial port on the JACE host, including its assigned COM address, there was no indication as to which ports were on option cards. Other details on option cards (and slots) were also unknown—for example, if an option slot was available (open) or not.

The Hardware Scan Service clarifies all of this by providing a complete hardware profile of the hosting JACE controller to its station and Workbench. Included is the controller’s “Product Model” type (combination of its base board and NPM module, if applicable) with a representative image, along with details about any installed option cards. This information can be useful when troubleshooting remotely, or even after adding an option card to confirm a COM address for a specific port.

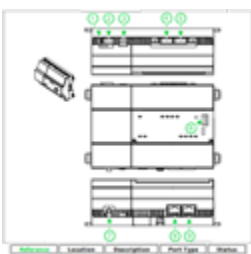
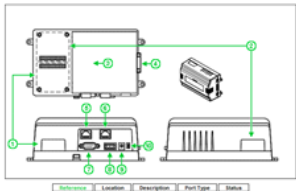
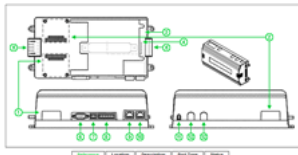
Additionally, this information could be useful to a developer of an “appliance” that runs on the controller, such that the appliance (station) configuration could logically adapt to different hardware profiles.

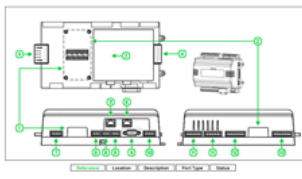
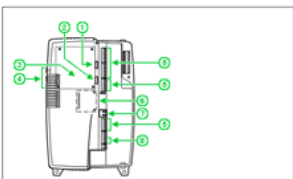
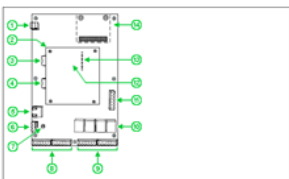
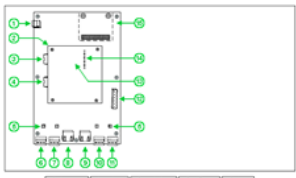
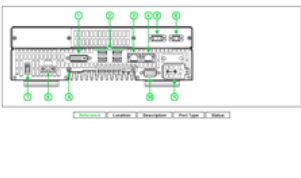
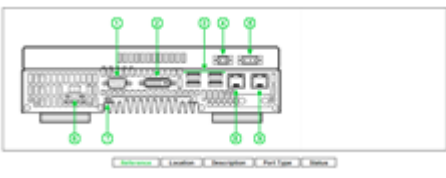
Note: Currently, the service is unaware of any attached I/O modules, including NDIO, NRIO, or Security types.

Parent topic: [JACE Hardware Scan Service](#)

Currently supported platforms

Currently supported platforms are listed below, showing a thumbnail of the “controller image” portion of the service’s default **Hardware Scan Service View**.

| Supported Platforms | | |
|---|---|--|
| JACE-8000 | JACE-2JACE-3EJACE-6JACE-6E | JACE-7(JACE-700) |
|  |  |  |
| Security Controllers (SEC-J-201 and SEC-J-601) | Controller-x02 Express (M2M JACE) | JACE-603 |

| Supported Platforms | | |
|---|---|--|
|  |  |  |
| JACE-645 | JACE-NXT | JACE-NXS |
|  |  |  |

Below any image in the **Hardware Scan Service View** is a hardware reference table with details corresponding to items with callouts. For more details, see “Hardware Scan Service View notes”.

Parent topic: [JACE Hardware Scan Service](#)

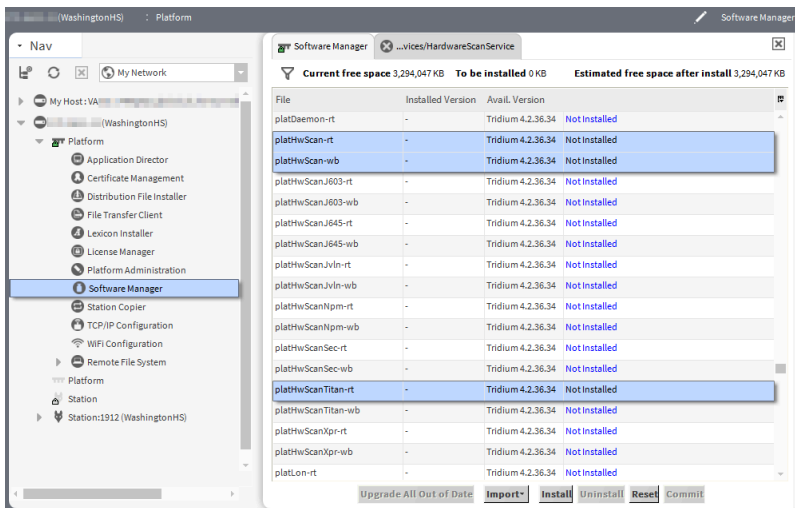
Adding the HardwareScanService

You can add the Hardware Scan Service in any JACE controller running AX-3.7 or later (including Niagara 4). To add the service, simply install the necessary software modules, using the **Software Manager** in a platform connection:

| For NiagaraAX installations | For Niagara 4 installations |
|-------------------------------|--|
| platHwScan (always necessary) | platHwScan-rt (always necessary)platHwScan-wb (always necessary) |
| platHwScanType | platHwScanType-rt |

For AX installations, the `Type` varies by the specific JACE model series (see table below). If you select only the `platHwScanType` module (such as `platHwScanNpm`), the required `platHwScan` module becomes automatically selected.

Note: Installing these modules on JACE controllers running AX results in a controller reboot. Installing the required modules on JACE-8000 controllers running Niagara 4 requires only that the station be restarted. Figure 1. Choosing the `platHwScan` modules needed for JACE-8000 running Niagara 4.



The image above shows the selected modules that are necessary for JACE-8000 controllers running Niagara 4.

The necessary platHwScanType is listed by controller series in the table below.

| Controller Series | platHwScanType module |
|---------------------------------------|-----------------------|
| JACE-8000 (running N4) | platHwScanTitan-rt |
| JACE-8000 (running AX-3.8U1) | platHwScanTitan |
| JACE-6, JACE-3E, JACE-6E | platHwScanNpm |
| JACE-7 | platHwScanJvln |
| JACE-603 (retrofit board) | platHwScanJ603 |
| JACE-645 (retrofit board) | platHwScanJ645 |
| JACE-x02 Express (202/602-XPR or M2M) | platHwScanXpr |
| JACE-NXT, JACE-NXS | platHwScanNx |
| SEC-J-601 | platHwScanSec |

Note: If you install an incorrect platHwScanType module, or install only platHwScan module, the default view on the station's Hardware Scan Service will simply display:

Jar file platHwScanType is required to support this platform.

In this case, simply install the correct type using the table above.

Parent topic: [JACE Hardware Scan Service](#)

Hardware Scan Service View notes

The main usage of the service is expected from its default view, the **Hardware Scan Service View**. Regardless of

the JACE controller type, this view consistently appears with one or more image views of the physical controller, with callouts described in a table below.

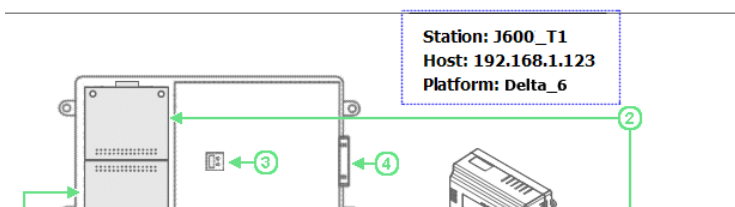
- **[Text in image notes](#)**
Three lines of text appear near the top of the controller image in a Hardware Scan Service View.
- **[Option card notes](#)**
Most JACEs support one or two option cards, which install in option slots on the controller's base board. An installed option card is "shaded gray" on the controller's image.
- **[Callout table notes](#)**
Callouts in the controller image are keyed to reference row entries in the table below it.
- **[Px usage of Hardware Scan Service View](#)**
The view is based on a Px file and graphic images inside platHwScan modules. If desired, you can make this view available on a Px view, by dragging the Hardware Scan Service onto a canvas pane.

Parent topic: [JACE Hardware Scan Service](#)

Text in image notes

Three lines of text appear near the top of the controller image in a Hardware Scan Service View.

Figure 1. Text in image notes in Hardware Scan Service View



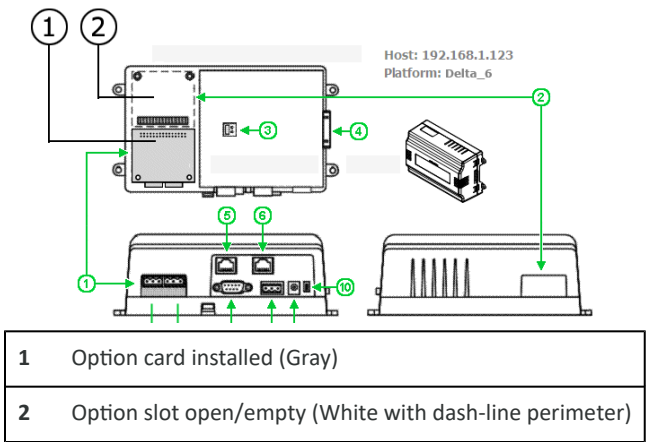
- **Station: Station_Name**
The name of the station running on the controller.
- **Host: IP_Address**
The IP address of the controller used in this station connection.
- **Platform: Product_Model**
"Product Model" descriptor, which may vary according to the vendor of the JACE controller.

Parent topic: [Hardware Scan Service View notes](#)

Option card notes

Most JACEs support one or two option cards, which install in option slots on the controller's base board. An installed option card is "shaded gray" on the controller's image.

Figure 1. Installed option cards are shaded gray



The numerical callout table below the diagram describes any installed option card. In the case of JACE-700, which has a MiniPCI slot in addition to two option card slots, any installed WiFi (MiniPCI) option is also indicated by gray shading.

Parent topic: [Hardware Scan Service View notes](#)

Callout table notes

Callouts in the controller image are keyed to reference row entries in the table below it.

Figure 1. Referenced items in table have defined data columns

| Reference | Location | Description | Port Type | Status |
|-----------|--------------|--------------------------------|-----------|-----------------------------|
| 1 | Option Slot1 | Dual Port RS-485 Option Card | | |
| 1-1 | Option Slot1 | COM3 | RS-485 | Owned by ModbusAsyncNetwork |
| 1-2 | Option Slot1 | COM4 | RS-485 | Owned by mstp2 |
| 2 | Option Slot2 | Single Port RS-232 Option Card | | |
| 2-1 | Option Slot2 | COM5 | RS-232 | Available |
| 3 | Base Unit | Serial Shell Jumper | | Serial Shell |
| 4 | Base Unit | I/O and Power Modules | NDIO | Available |

By default, columns in this table are labeled as below, and are explained as follows:

- Reference

The callout number for the item in the controller image. If an option card port, it may use a “1-n” or “2-n” number that relates to option slot 1 or 2 (for those platforms with two option card slots).

- Location

Usually either “Base Unit”, “Option Slot1”, or “Option Slot2” (the callout line in the image clarifies).

- Description

Text description of the item, whether option card, option module, COM port, or other physical feature.

Note: Any “vendor-unique” option card may incorrectly list, e.g. “Single Port RS-232 Option Card”. We

recommend that you verify with the option card’s vendor as to how it lists/appears in this service.

- **Port Type**

If applicable, the type of controller port, e.g. RS-485, RS-232, Ethernet, NDIO (Niagara Direct Input Output), and so on. If not applicable to the referenced item, this field is blank.

- **Status**

If applicable, the disposition of the referenced item. If not applicable, this field is blank.

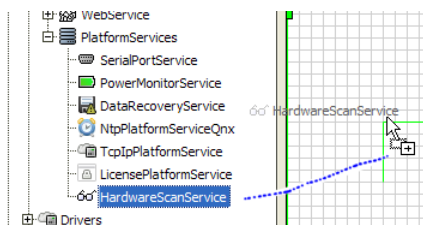
- If a serial port currently in use in the station, status reads “Owned by entity”, where the entity could be a driver network (say `ModbusAsyncNetwork`), or low-level driver (say `mstp1`, for an `MstpPort` in a `BacnetNetwork`), or even `Serial Shell` for a COM1 port—if the serial shell jumper is installed. If a serial port not currently used in the station, status is `Available`.
- If an Ethernet LAN port, status is either `Available` (port enabled in controller’s TCP/IP configuration) or `Disabled` (port not enabled in controller’s TCP/IP configuration).
- Platforms with a “serial shell jumper” have status either `Normal Operation` or `Serial Shell`.

Parent topic: [Hardware Scan Service View notes](#)

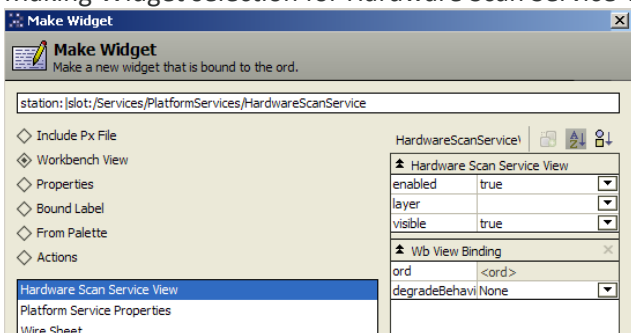
Px usage of Hardware Scan Service View

The view is based on a Px file and graphic images inside `platHwScan` modules. If desired, you can make this view available on a Px view, by dragging the Hardware Scan Service onto a canvas pane.

Figure 1. Dragging the HardwareScanService onto a Px view



Making Widget selection for Hardware Scan Service View



If doing this, in the resulting “Make Widget” popup dialog select the **Workbench View** and **Hardware Scan Service View**, as shown above.

You must resize the copied widget to see all of the view. To do this, enter the correct pixel dimensions for the object by editing its properties in the **PxEditor Widget Tree** (right-click on **HardwareScanServiceView** > **Edit Properties**). Depending on the controller series, the overall “footprint” (pixel dimensions) varies. The approximate width and height (in pixels) for each view, including the lower “hardware reference table” area plus side scroll bar, are listed in the following table.

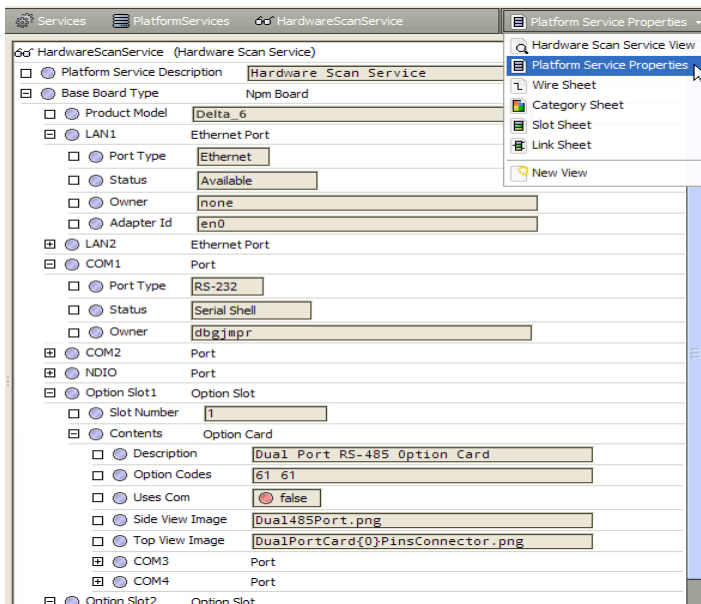
| Controller Series | Hardware Scan Service View Width x Height (pixels) |
|--|---|
| JACE-8000 | 640 x 987 |
| JACE-6, JACE-3E, JACE-6E | 660 x 720 |
| JACE-7 | 820 x 700 |
| JACE-603 or JACE-645 (403/545 with retrofit board) | 660 x 740 |
| JACE-NXT | 830 x 600 |
| JACE-NXS | 800 x 600 |
| SEC-J-601 | 790 x 800 |

Parent topic: [Hardware Scan Service View notes](#)

HardwareScanService properties

The HardwareScanService has a number of properties, available by selecting its property sheet from the view selector, as shown. Values from many of these properties are reflected in the graphical default view.

Figure 1. Property sheet for example HardwareScanService



All properties but one are children of the “Base Board Type” container, which varies by controller type. The immediate child property is “Product Model”, a string property that describes the controller model. Other children are containers that contain a mixture of other string properties and enumerated values, such as for “Status” or “Port Type”. If needed, you can link any of these properties into other station logic.

As with most other station platform services, there is an available “Poll” action on the HardwareScanService. However, usage should rarely be needed, due to the static nature of hardware configuration.

Parent topic: [JACE Hardware Scan Service](#)

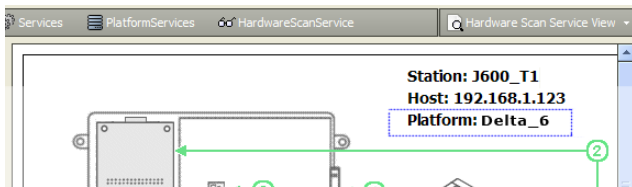
Lexicon customizing of HardwareScanService

You can customize text descriptors and values that appear in the service’s **Hardware Scan Service View** by editing the lexicon for the `platHwScan` module and various `platHwScanType` modules.

For example, if you edit the following lexicon key for the `platHwScanNpm` module as shown:

```
NPM6=Delta_6
```

The “Platform=” value near the top of the controller image will display the brand name, “Delta_6” (instead of the default product model name) when viewing the service in the controller, as shown below.



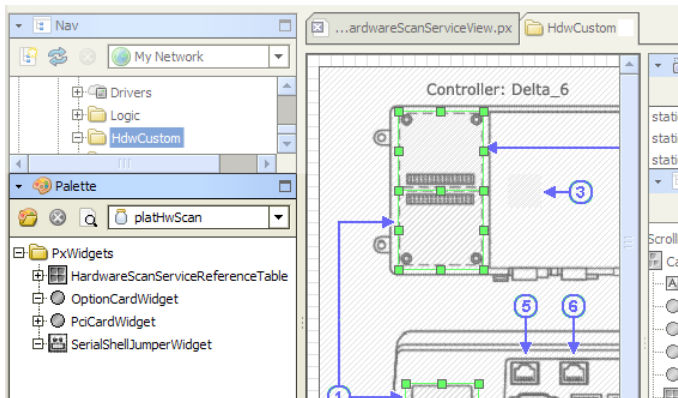
Note: Before such changes can become effective, you need to save and install the lexicon changes in the target JACE, and also reboot that controller.

Parent topic: [JACE Hardware Scan Service](#)

Px customization

You can make a Px view in a station that provides a customized alternative to the default **Hardware Scan Service View**, by using Px widgets available in the palette of the `platHwScan` module, as well any custom edited image (.png) file or files (for the controller “layout” diagrams).

Figure 1. Palette of `platHwScan` module has four Px widgets



As shown here, there are four types of Px widgets in the `platHwScan` palette:

- HardwareScanServiceReferenceTable

This widget automatically receives values from the station's platform HardwareScanService, presenting it in a table with five columns. It applies to any supported controller type.

- **OptionCardWidget**

This widget represents either a top view or side (port) view of any installed option card, and is designed to overlay an image file that represents a top and/or side layout view of a controller type. It applies to all supported controller types except the Windows-based ones (JACE-NXT, JACE-NXS).

Note: An OptionCardWidget must have its Slot property value set to either 1 or 2, depending on location. If copied from the palette, the slot property value defaults to 0.

- **PciCardWidget**

This widget represents an installed MiniPCI card, and also overlays a controller layout image file. It applies only to a WiFi option for JACE-7 (700) series controller (using platHwScanJvln module).

- **SerialShellJumperWidget**

This widget represents the current "serial shell jumper" position of a controller, and also overlays a controller layout image file. It applies to the same controller types as the OptionCardWidget.

Configurable properties for these Px widgets include:

- Visible: true (default)/false
- Enabled: true (default)/false
- As well as several properties that allow you to configure the object's appearance.

Note: The different platHwScanType modules may also contain Px widgets specific to the type of controller. For example, platHwScanTitan contains the OptionModuleWidget.

- [Px widget usage in platHwScan module](#)

Px widgets are used in the default **Hardware Scan Service View**. A good way to see how is to examine the local modules on your PC installation. Do this by expanding a platHwScanType module to look at the contents of its px folder in the Px Editor (edit mode).

- [Example — customized Px view for Hardware Scan Service](#)

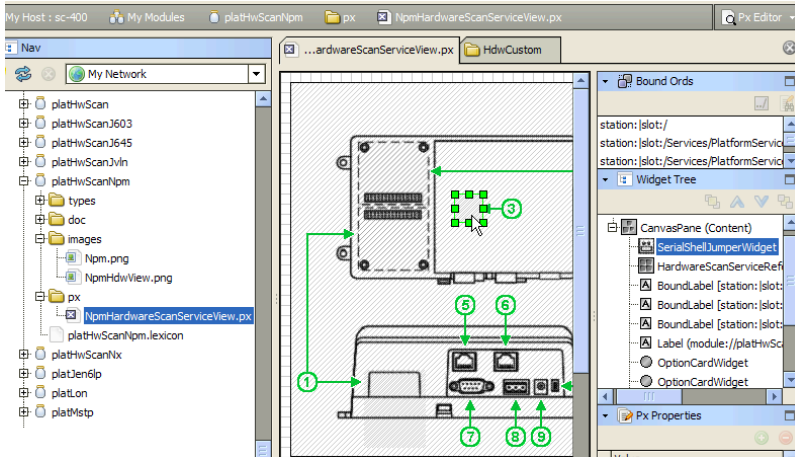
Shown below is a customized Px view for the following controller series: JACE-6, JACE-3E, JACE-6E.

Parent topic: [JACE Hardware Scan Service](#)

[Px widget usage in platHwScan module](#)

Px widgets are used in the default **Hardware Scan Service View**. A good way to see how is to examine the local modules on your PC installation. Do this by expanding a platHwScanType module to look at the contents of its px folder in the Px Editor (edit mode).

Figure 1. Looking at the Px view in the platHwScanNpm module



Look at widget properties to see how layout parameters and other items are set.

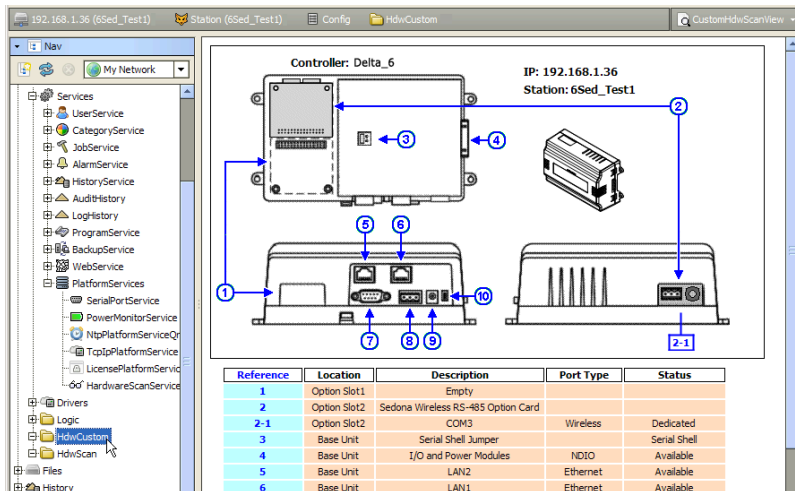
Parent topic: [Px customization](#)

Example — customized Px view for Hardware Scan Service

Shown below is a customized Px view for the following controller series: JACE-6, JACE-3E, JACE-6E.

In this example, the controller image (NpmHdwView.png) was copied from the platHwScanNpm module and then edited in a graphics program to change the color of the callout arrows and numbers from the default green to blue. The edited png file is then copied to the controller's station folder (using the platform **File Transfer Client** view), and then referenced in the BoundLabel widget for it in the Px view.

Figure 1. Example customized Px view for Hardware Scan Service



The OptionCardWidgets used in the “side view” positions (topView=false) had formatPalette child properties for borderColor and referenceColor changed from (the default) green to blue, to match the edited image callout lines. (The color of these callout lines and border is set in the widgets).

Note: An OptionCardWidget must have its “slot” property value set to either 1 or 2, depending on location. If copied from the palette, the slot property value defaults to 0.

In the same way, properties were edited for the `HardwareScanServiceReferenceTable` widget, including reducing `columnGap` and `rowGap` from 3.0 to 2.0. Also this widget's `formatPalette` had its child property for `referenceColor` set from (the default) green to blue, and properties `referenceBackgroundColor` and `dataBackgroundColor` were set from (default) white to colors cyan and peach, respectively.

Finally in this example, the `BoundLabel` widgets for "Platform:" and "Host:" near the top of the view had text property edits, to read (instead) "Controller:" and "IP:", and all three `BoundLabel` widgets were repositioned and reduced in font size from 14pt to 12pt.

Note: You can save time by copying the contents of a Px file from one of your local `platHwScanType` modules (for the appropriate controller type), then pasting that in as the source XML for your new Px view. Using this as a starting point, you can then use the Px Editor to make tweaks in widget properties, and/or reference images outside of the `platHwScanType` modules. Combined with changes in lexicons for the `platHwScan` modules, this can provide a branded view of the Hardware Scan Service information.

Parent topic: [Px customization](#)

Using the dashboard feature

The Dashboard feature enables authorized users to edit a limited set of graphics without having to use the Px Editor, effectively customizing the view as desired. Additionally, each user can save or clear changes he or she has made to the dashboard without affecting another user's changes. For example, a Systems Integrator can add a Dashboard Pane to a Px page which sets up the default configuration for the dashboard. Another user subsequently viewing that dashboard, such as a Facilities Manager who is not PxEditor-trained, is able to customize the dashboard by dragging additional components (from a hierarchy or the Nav Tree) onto the WebWidget displayed in the dashboard without using PxEditor, and save those changes. When each user reopens the dashboard they see only their own saved customizations. The functionality described here works in Workbench as well as in the HTML5 Hx profile.

Currently, there are two out-of-the-box WebWidgets that you can use in a dashboard: CircularGauge and Chart from the **webChart** palette. Both of these widgets allow you to dynamically drag and drop applicable components onto them so the new values are visible instantly. Combining these widgets with a dashboard allows a user to make changes, save the changes, and show their preferred content by default.

This functionality allows multiple users to personalize a PxView as needed without altering the original view. It also eliminates the need to use PxEditor to create several variants of the same PxView to accommodate different users.

Requirements

In order to use dashboards you must install the following modules: dashboard-rt, dashboard-ux, and dashboard-wb.

You must install the **DashboardService** in the running station.

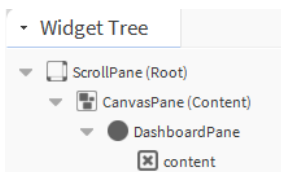
Attempting to create a dashboard without first installing the DashboardService, causes an error which alerts you to add the service to the station. Also, the DashboardPane will display an error if the DashboardService is not in an operational status (i.e. disabled).

- [Setting up a dashboard](#)
Using the **PxEditor** view, you can setup a dashboard by adding a **Dashboard** Pane to the Px page and add an HTML5 "dashboardable" WebWidget (WebChart, CircularGauge, or a custom WebWidget). This procedure describes adding the Chart WebWidget to a Px page.
- [Editing a dashboard](#)
When viewing a dashboard you can dynamically drag and drop applicable components onto the dashboard and see new values displayed instantly. This procedure describes adding a BooleanWritable point to a dashboard displaying a WebChart that is already plotting data for a NumericWritable point.
- [Clearing your dashboard data](#)
If you have changed data when viewing dashboards, there are a couple of ways to clear your changes. One method, invoked with the dashboard **Clear** command, clears your saved changes from a dashboard while you are viewing it. The other method, invoked with an action on the service, clears only changes that you have made from all of the dashboards in the system. Both methods effectively reset those dashboards to the original (default) data.
- [Clearing all dashboard data](#)
You can invoke this action to clear **all** changes (by any user) that have been made to dashboards in the system. This reset all dashboards in the system to the original (default) data.
- [Dashboards concepts](#)
The dashboard framework is used to write dashboard-specific data to and from a station for a specific user.

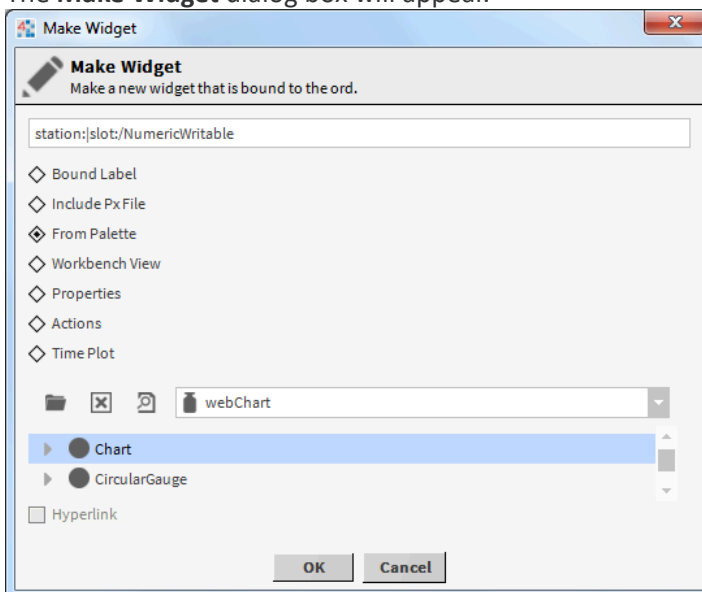
Setting up a dashboard

Using the **PxEditor** view, you can setup a dashboard by adding a **Dashboard** Pane to the Px page and add an HTML5 “dashboardable” WebWidget (WebChart, CircularGauge, or a custom WebWidget). This procedure describes adding the Chart WebWidget to a Px page.

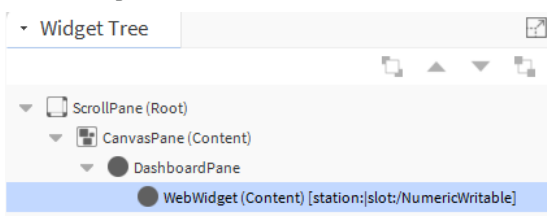
- **DashboardService** added to station **Services** node
 - Px file open in the PxEditor mode in Workbench
 - **dashboard** and **webChart** palettes open
1. From the **dashboard** palette, drag the DashboardPane component onto the CanvasPane in the Px page.
 2. In the Px page Widget Tree, expand the DashboardPane to display the “content” subproperty, as shown.



3. From the Nav tree, drag a point (such as a NumericWritable) directly onto the content space under the DashboardPane in the widget tree.
The **Make Widget** dialog box will appear.



4. Select From Palette from the options and select Chart from the **webChart** palette, and click **OK**. In the Widget Tree under the DashboardPane, the content subproperty displays as:
WebWidget (Content) [station:|slot:/NumericWritable].



This configures the default Ord value for this dashboard. The web chart immediately begins plotting data for the point.

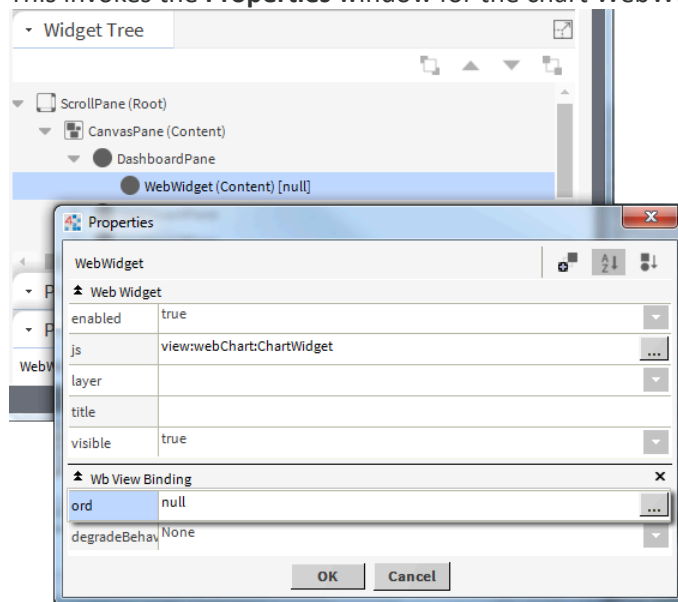
5. In the Workbench toolbar, click **Save**.
6. Switch to **PxView** mode (click **PxEditor** > **Toggle View/Edit Mode**).

Note: A Dashboard Pane may contain only one WebWidget. If you wish to include several dynamic WebWidgets on your Px page, you must add a Dashboard Pane for each WebWidget.

The web chart in the **Px View** plots live data (and possibly historic data) for the selected point. Since the view features a dashboard, you can dynamically edit the data shown.

Another way to add the Chart WebWidget to a dashboard is by dragging the Chart widget from the **webChart** palette, to the DashboardPane's content property in the Widget Tree. Double-click on WebWidget (Content) [null].

This invokes the **Properties** window for the chart WebWidget showing a null value in the Ord slot, as shown.



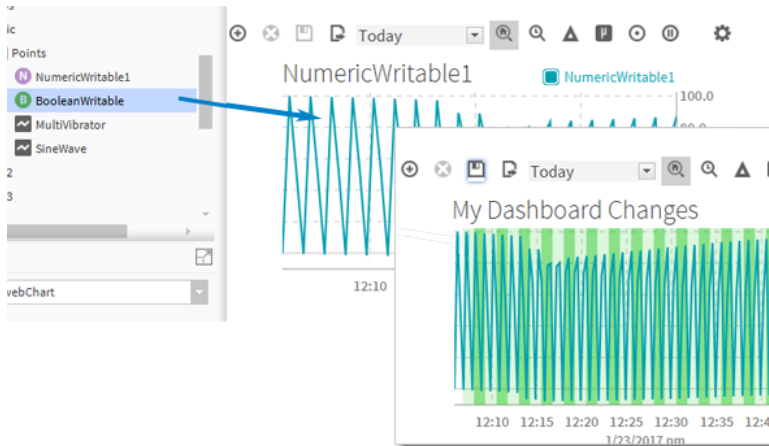
Click on the selection button at the right of the Ord field and use the Component Chooser to navigate to the point whose data you desire to plot on the chart, and click OK.

Parent topic: [Using the dashboard feature](#)

Editing a dashboard

When viewing a dashboard you can dynamically drag and drop applicable components onto the dashboard and see new values displayed instantly. This procedure describes adding a BooleanWritable point to a dashboard displaying a WebChart that is already plotting data for a NumericWritable point.

- An existing Px page with a dashboard viewed either in the Workbench PxView mode or in a browser connection. The dashboard displays a WebChart that is already plotting data for a NumericWritable point
1. In the PxViewer or a browser, change the dashboard data by dragging another point (for example, a BooleanWritable) from the Nav Tree onto the chart. The dashboard content immediately changes to display data from both points, as shown. Also, editing data in a dashboard activates the **Save** command.



2. Click the **Save** command at the top of the dashboard.
Once saved, your changes will load the next time you view this dashboard.

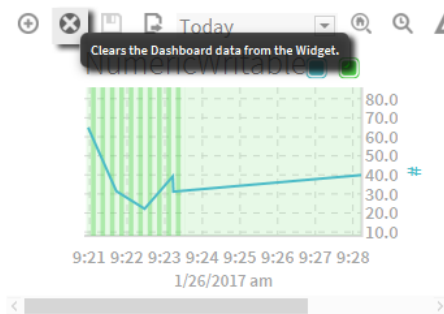
Note: Saving a change activates the **Clear** command so that you can remove your changes if you wish.

Parent topic: [Using the dashboard feature](#)

Clearing your dashboard data

If you have changed data when viewing dashboards, there are a couple of ways to clear your changes. One method, invoked with the dashboard **Clear** command, clears your saved changes from a dashboard while you are viewing it. The other method, invoked with an action on the service, clears only changes that you have made from all of the dashboards in the system. Both methods effectively reset those dashboards to the original (default) data.

- Logged on with admin invocation privileges on the **DashboardService**
 - One or more existing dashboards with changes that you have made.
1. To clear your changes from a single dashboard while you are viewing it:
 - a. Click the icon for the **Clear** command, as shown here.



Note: The CircularGauge WebWidget displays the same **Clear** and **Save** commands as those displayed in the Chart WebWidget.

- b. In the confirmation popup window, click **Yes** to clear your changes.
This option clears only changes you have made to this dashboard, resetting the dashboard to the default data.
2. To clear your changes from all dashboards on the system:
 - a. Right-click on the **DashboardService** and click **Actions > Clear My Dashboards**.

- b. In the **Clear My Dashboards** dialog, click **Yes** to clear your changes.

This action clears only your changes from all of the dashboards on the system by removing the associated dashboard files (db*.json) stored within the station's protected space.

Note: If viewing a dashboard (showing your changes) when you select this option, you may need to click **Refresh** in the Workbench toolbar in order to see that your changes have been cleared.

Parent topic: [Using the dashboard feature](#)

Clearing all dashboard data

You can invoke this action to clear **all** changes (by any user) that have been made to dashboards in the system. This resets all dashboards in the system to the original (default) data.

- Logged on as a super user

CAUTION: Be aware that invoking this action removes dashboard customizations made by all users of this system.

1. Right-click on DashboardService and select **Actions > Clear All Dashboards**
2. In the **Clear All Dashboards** confirmation dialog, click **Yes** to clear all changes.

Note: If viewing a dashboard (showing changes) when you select this option, you may need to click Refresh in the toolbar in order to see that the changes have been cleared.

This action clears all changes from all of the dashboards on the system by removing the associated dashboard files (db*.json) stored within the station's protected space.

Parent topic: [Using the dashboard feature](#)

Dashboards concepts

The dashboard framework is used to write dashboard-specific data to and from a station for a specific user.

For example, say you have a Px page with a Web Chart on it. Another user (possibly not trained to use PxEditor) would like to add more data to the chart and save those changes so that when the user navigates to the page again, the saved chart modifications are visible. Also, if that user logs on from a different device those saved chart modifications are visible.

Note when other users log on, if they have not made changes they see the web chart in the original form. They do not see another user's customized chart data.

Also note, dashboard widgets update with live data.

When a dashboard WebWidget is modified and saved, a dashboard file is created to store the data since it cannot be stored within the Px file. If a dashboard file does not exist, one is created. The dashboard file is stored within the station's protected space, with the .json file extension.

- [Dashboard Service](#)
The **DashboardService** must be added to a station's **Services** container. Note that using dashboards requires that the dashboard-rt, dashboard-ux, and dashboard-wb modules be installed on the platform. The service is responsible for reading dashboard data from, and writing dashboard data to db*.json files stored in the dashboards directory which is located within the protected station home (local:|platformssl:|file:~stations/stationName/dashboards/userName).

- **[Dashboard pane](#)**
A DashboardPane is a container for a single WebWidget that can be configured for use in a dashboard. Add the pane to a Px page via the PxEditor. Only HTML5 WebWidgets can be used in a DashboardPane. Specifically, the WebChart and CircularGauge widgets. Typically a dashboard is comprised of multiple DashboardPanes, each containing a WebWidget.
- **[Dashboard commands](#)**
Commands available when viewing a dashboard are **Clear** and **Save**, as shown.
- **[Frequently asked questions](#)**
This topic provides frequently asked questions and answers about dashboards, the feature which enables you to change and edit a limited set of your graphics without using PxEditor.

Parent topic: [Using the dashboard feature](#)

Dashboard Service

The **DashboardService** must be added to a station's **Services** container. Note that using dashboards requires that the dashboard-rt, dashboard-ux, and dashboard-wb modules be installed on the platform. The service is responsible for reading dashboard data from, and writing dashboard data to db*.json files stored in the dashboards directory which is located within the protected station home (local:|platformssl:|file:~stations/stationName/dashboards/username).

Actions

The **DashboardService** provides two admin actions that permit clearing your own dashboard data from the system, and if logged on as a super user, you can clear all dashboard data (made by any user) from the system. Right-click on the service to select either of these actions:

- **Clear My Dashboards:**

Clears all of the dashboard data (changes you have made) from the system. Therefore, if you are logged on (and have admin invocation privileges on the service), you can invoke this action to clear all of your dashboard data from the system.

- **Clear All Dashboards:**

If you are logged on as a super user, you can invoke this action to clear dashboard data for all users from the system. If not logged on as a super user, you will see an error message upon invoking this action.

| Name | Value | Description |
|-------------|-----------|---|
| Status | read-only | Reports the condition of the entity or process at last polling. {ok} indicates that the component is licensed and polling successfully. {down} indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. {disabled} indicates that the Enable property is set to false. {fault} indicates another problem. Refer to Fault Cause for more information. |
| Fault Cause | read-only | Indicates the reason why a system object (network, device, component, extension, |

| Name | Value | Description |
|---------|---------------|--|
| | | etc.) is not working (in fault). This property is empty unless a fault exists. |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |

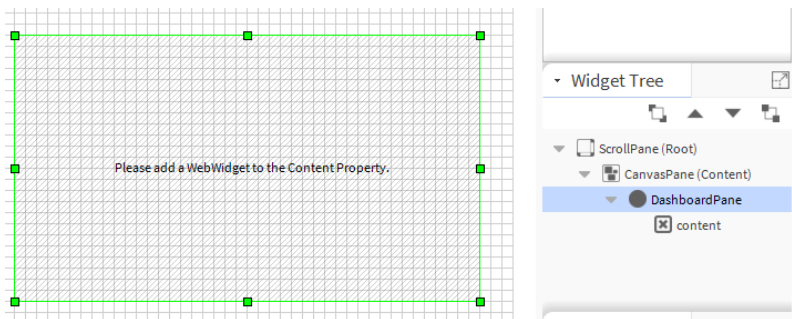
Parent topic: [Dashboards concepts](#)

Dashboard pane

A DashboardPane is a container for a single WebWidget that can be configured for use in a dashboard. Add the pane to a Px page via the PxEditor. Only HTML5 WebWidgets can be used in a DashboardPane. Specifically, the WebChart and CircularGauge widgets. Typically a dashboard is comprised of multiple DashboardPanes, each containing a WebWidget.

Dashboard Pane properties

Figure 1. Dashboard Pane



A dashboard is constructed on a Px page. After adding the DashboardService to the station, you can drag and drop a DashboardPane multiple times from the dashboard palette.

Note: If a DashboardPane is added to a Px page without the DashboardService being added to the station, a warning message alerts you. Also, a warning message alerts you if the DashboardService is disabled.

| Property | Value | Description |
|----------|--|---|
| Visible | true (default), false | Enables/disables display of the widget. |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Id | %ord%;%viewId%;%widgetOrd%;%jsOrd% (default) | This BFormat string uniquely identifies the dashboard data for a particular user. Although, it can be changed if required, it is not recommended since this is an advanced feature. |
| Layout | | Unique position and dimensions of the |

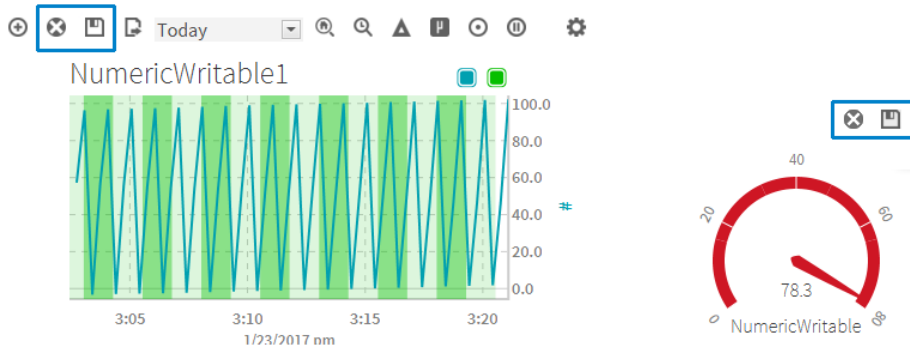
| Property | Value | Description |
|----------|-----------|--|
| | | widget on the Px page. Includes X and Y coordinates, width in pixels, height in pixels. |
| Content | WebWidget | <p>A WebWidget is required in this property, for example: CircularGauge or WebChart found in the webchart palette.</p> <p>If a DashboardPane's Content slot is empty, by default a warning notifies you of the need to add a WebWidget to the Content property.</p> <hr/> <p>Tip: You must use the Widget Tree view in the Px Editor to expand the DashboardPane so that you can drag and drop a WebWidget onto the Content subproperty.</p> <hr/> |

Parent topic: [Dashboards concepts](#)

Dashboard commands

Commands available when viewing a dashboard are **Clear** and **Save**, as shown.

Figure 1. Clear and Save dashboard commands



- **Save:**

Changing any property in a running dashboard enables the **Save** command. When you click **Save**, the framework saves the dashboard.

- **Clear Dashboard:**

The saved dashboard data being loaded into the widget, enables the **Clear Dashboard** command. If invoked, this clears any added dashboard information from this widget only. After clearing the data, the page reloads. If you attempt to run this command when there are changes that still need to be saved, an alert message appears.

For details on available actions see, DashboardService, Actions.

Parent topic: [Dashboards concepts](#)

Frequently asked questions

This topic provides frequently asked questions and answers about dashboards, the feature which enables you to change and edit a limited set of your graphics without using PxEditor.

Q: How do I create a dashboard?

A: Create a Px page. For each widget that you want to dashboard, add a DashboardPane from the dashboard palette to contain it.

Does each Widget on a Px page require a DashboardPane?

A: No. Only the parts of the Px page that are intended to be “dashboardable” (dynamically modified by the user) require a parent DashboardPane. All other widgets can be added to the Px page as normal.

Q: Which WebWidgets can be used on a Dashboard?

A: Dashboard supports only the HTML5 bajaux WebWidgets: CircularGauge and Chart, but custom WebWidgets may be built to take advantage of dashboard features.

Q: If I log in as the same user in a different browser, will I get the same Dashboard?

A: Yes you will. The information held on Dashboards is held for a specific user. If you log on as the same user, you'll get the same Dashboard data regardless of what browser you use (Chrome, Workbench, or otherwise).

Parent topic: [Dashboards concepts](#)

BACnet and Niagara 4

The user interface for configuring and managing a BACnet network is the same under Niagara 4 as it was under previous versions of Niagara. This chapter describes usage of the new bacnetUtil module.

A new module, BACnetUtil, has been added for troubleshooting connections. [BACnet troubleshooting](#) begins the information about BACnetUtil.

- [BACnet troubleshooting](#)
This topic summarizes things to do to resolve certain BACnet networking problems.
- [Reference](#)
The topics that follow provide detailed documentation for each component, plugin (view) and window that supports this system feature.

BACnet troubleshooting

This topic summarizes things to do to resolve certain BACnet networking problems.

When I start a station the BACnet/Ethernet Port goes into fault with the message, “Verify WinPcap 4.1.x is installed.”

WinPcap must be manually installed to use BACnet/Ethernet on supported Niagara 4 Windows platforms. Download the utility from <https://www.winpcap.org/install/>, install it, and restart the station.

I have connected all network devices as described in the documentation, but Workbench fails to discover any devices.

Failure to discover may be due to:

- duplicate mac addresses
- mis-configured baud rates
- a max master that is too small

Use bacnetUtil to diagnose the above problems.

How can I diagnose intermittent MS/TP network problems

The BTokens component may be the most useful MS/TP troubleshooting tool. This component provides visibility into the health of an MS/TP network at a new level. Previously this type of information could only be gathered by observing and interpreting the blink pattern of the device’s serial activity LEDs. Knowing the bus is healthy at the token passing level may help shift the focus of an investigation to a remotely solvable configuration problem that does not require dispatching a technician to the site to investigate.

Initial offsite diagnosis of intermittent problems can be performed using the BTokens component, standard Niagara point extensions and web charts. For example, you can detect interruptions in bus communications by graphing several relay values on the same WebChart as the tokens per second and monitoring the bus health.

How can I improve performance of older devices

Device overrides can be a useful tool for managing performance limitations in some devices. For example, older devices in the field may claim support for features like “readPropertyMultiple” (answering multiple questions per request), but do not do a great job of answering large multiple question requests in a timely manner. Those older devices may be able to more quickly respond to 30 separate requests containing only one question, than

one request containing 30 questions. Adding an rpmOverride to the device may improve overall performance of communication with the device by instructing Niagara to send only single question requests to the device.

- [Diagnosing network problems with metrics](#)
bacnetUtil provides tools to monitor the health of the BACnet ports in a station, and diagnose port problems. Three **bacnetUtil** tools count tokens and messages.
- [Setting up a wiretap](#)
Wiretaps evaluate (sniff) messages after they have been processed by a network port. You add wiretaps as children of the **NetworkPorts** container in the same way that you add metric components.
- [Overriding a device](#)
Occasionally, BACnet devices report false capabilities or external forces make operating a device with the provided values impossible. **Niagara** provides override components that you can add as children of a **BBacnetDevice**. These components provide a persistent mechanism to ignore device-provided values.

Parent topic: [BACnet and Niagara 4](#)

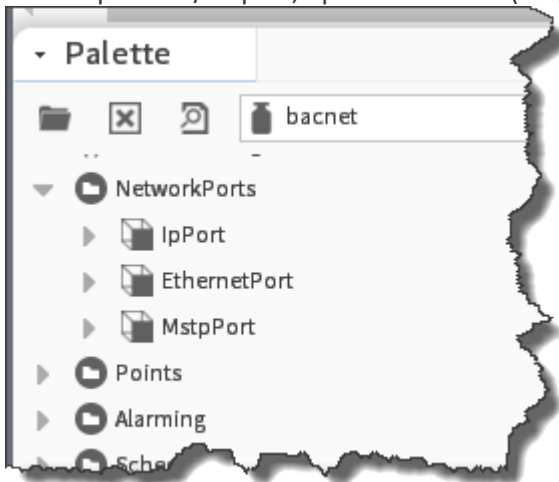
Diagnosing network problems with metrics

bacnetUtil provides tools to monitor the health of the BACnet ports in a station, and diagnose port problems. Three **bacnetUtil** tools count tokens and messages.

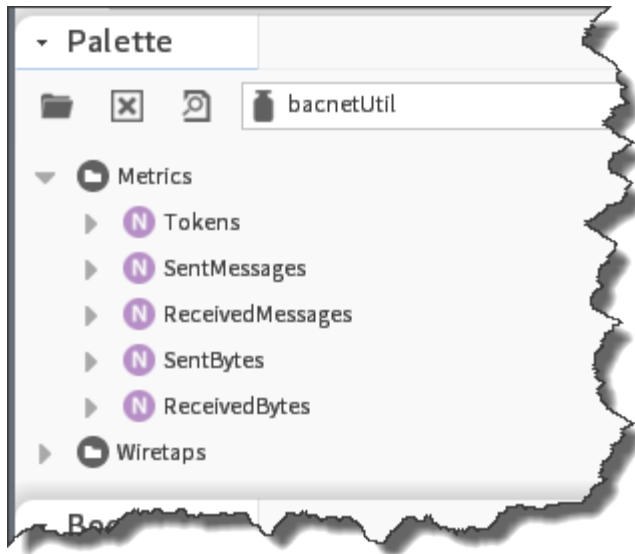
Your BACnet network has been set up and all devices are configured and communicating.

Some networks do not allow the Wireshark utility to be installed on their control networks. Investigating certain BACnet communication problems without proper network diagnostic tools can be difficult. The **bacnetUtil** module is intended to provide diagnostic resources for these networks.

1. To prepare the Nav tree, expand **Config > Drivers > BacnetNetwork > Bacnet Comm > Network**.
2. To set up an MS/TP port, open the **bacnet** (Niagara BACnet Driver) palette and expand **NetworkPorts**.



3. Drag the **MstpPort** container to the **Network** container in the Nav tree.



4. To add metrics, open the **bacnetUtil** palette, expand the **Metrics** container in the palette, and drag the Tokens, SentMessages and Received Messages components to the **MstpPort** node in the Nav tree. This starts the system monitoring tokens, sent and received messages.
5. To view a chart, right-click the component (Tokens, SentMessages or Received Messages) and click **Views > Chart**. It may take a few moments for the chart to appear.

- **[Token metrics](#)**

The Tokens component aids in diagnosing certain configuration problems. While the problems may require a site visit to resolve, the information gathered should aid in diagnosis and repair.

- **[Collecting data on MS/TP ports](#)**

MS/TP statistics running for MS/TP ports are helpful diagnostic tools. The **bacnetUtil** module provides a **Tokens** component that allows you to collect data for troubleshooting purposes and to remotely control the health of **MstpPorts** in a controller. You can add the **Tokens** component as a child of a controller's BACnet MS/TP port.

- **[NetworkPort metrics](#)**

Counting the number of messages sent and received per second is straightforward. The **SentMessages** and **ReceivedMessages** components perform as intuitively expected.

- **[Device metrics](#)**

Knowing the latency of a device can be quite useful when diagnosing individual network devices that may be responding slowly, but not slowly enough to trigger an APDU timeout (APDU - Application layer Data Unit).

Parent topic: [BACnet troubleshooting](#)

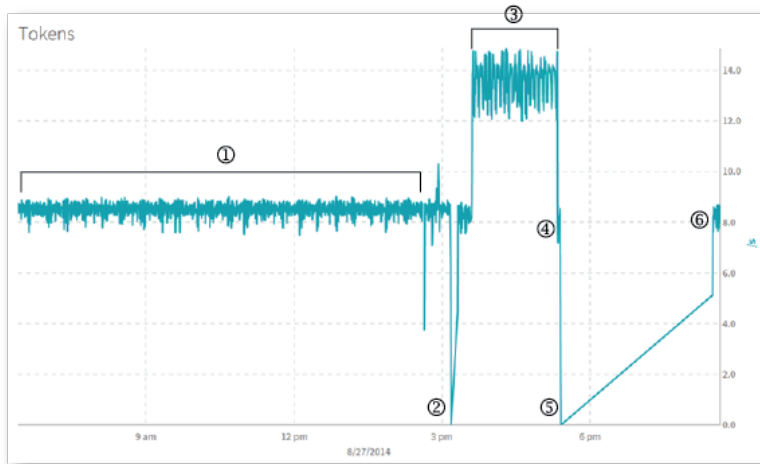
Token metrics

The Tokens component aids in diagnosing certain configuration problems. While the problems may require a site visit to resolve, the information gathered should aid in diagnosis and repair.

The Tokens per second (TPS) metric reports the tokens generated by the device per second. The number of tokens generated may seem backwards at first glance because, a trunk with many devices, passes *fewer* tokens per second. The fewer the devices on the trunk, the more tokens generated.

There is no rule-of-thumb for tokens per second, no good or bad. Tokens per second varies wildly based on device count, vendor implementation, baud rate, trunk utilization, and more. This limits the use of the metric to viewing changes over time. You need to establish a baseline for each particular trunk, and then investigate any deviations from its unique norm.

The following is an example of a couple of inflection points on a graph.



These are the types of problems that this graph can help you investigate.

1. Notice that from 7 am until 2:45 pm the tokens per second indicates a healthy MS/TP trunk, passing tokens at a constant rate. It is impossible to see the effect of applications messages at this level, as repetitive applications messages cause a repetitive impact on TPS.
2. Each spike or deviation was the result of adding or removing devices to or from the trunk. When the TPS initially dropped to zero (0) shortly after 3 pm, the controller completely stopped passing tokens. In this case, the MS/TP connector was unplugged from the controller.
3. The steady increase to ~13 TPS occurred when two other devices were removed from the trunk.
4. Shortly thereafter, the two devices were restored to the network and the TPS returned to its previous level (8).
5. The TPS again dropped to zero (0) when the controller was again unplugged.
6. Once the controller was plugged back in again, things returned to the normal 8 TPS. (The angle of the line is a COV (Coefficient of Variation) history artifact, the value was zero (0) until it was five (5)).

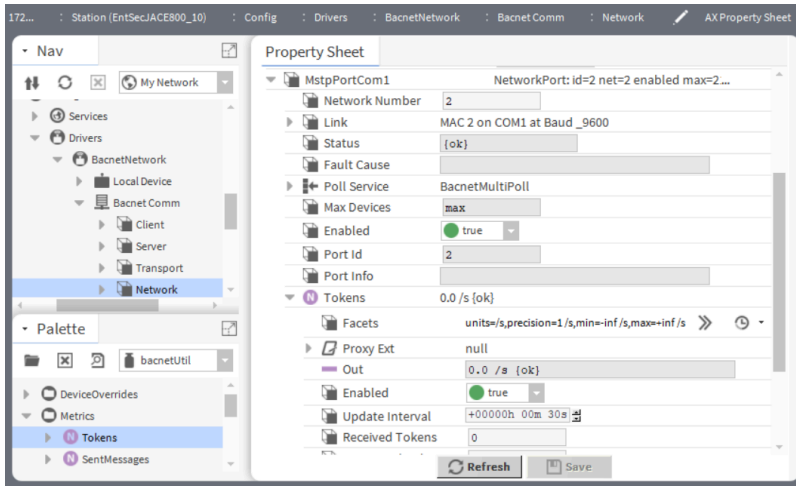
This sequence of events represents only the tip of the iceberg of possibilities. When Parse All Properties is set to true, the **Tokens** component exposes all of the current properties and any that may be added in the future.

Parent topic: [Diagnosing network problems with metrics](#)

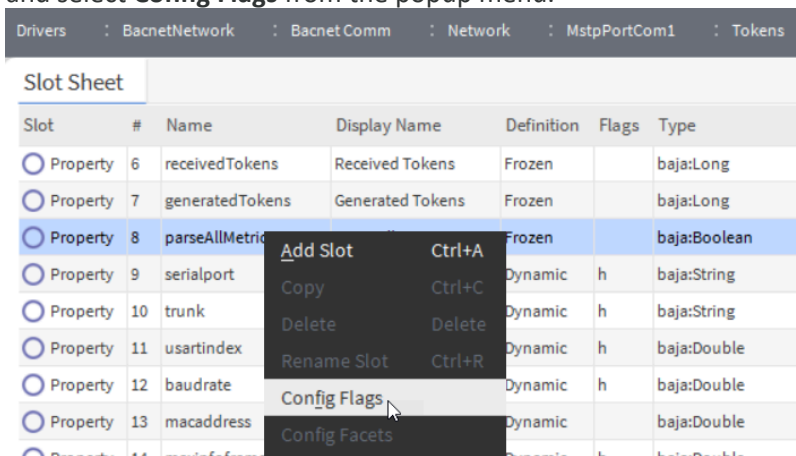
Collecting data on MS/TP ports

MS/TP statistics running for MS/TP ports are helpful diagnostic tools. The bacnetUtil module provides a **Tokens** component that allows you to collect data for troubleshooting purposes and to remotely control the health of **MstpPorts** in a controller. You can add the **Tokens** component as a child of a controller's BACnet MS/TP port.

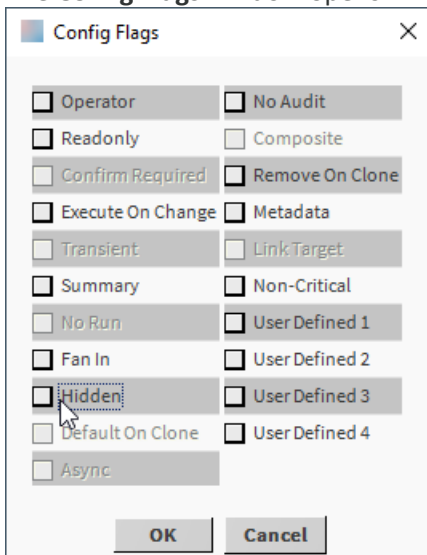
- In your controller you have expanded **Config > Drivers > BacnetNetwork > Bacnet Comm > Network**.
 - You have added a **MstpPort** to your Bacnet network.
1. To add the **Tokens** component, open the bacnetUtil palette, and drag the **Tokens** component from the palette to the respective **MstpPort**.



- To view the token properties of interest, unhide them. To do so, right-click the **Tokens** component under the MstpPort, select **ViewsAX Slot Sheet**, right-click the token property that you want to make visible and select **Config Flags** from the popup menu.



The **Config Flags** window opens.



- Deselect the **Hidden** checkbox and click **OK**.

Note: When the Parse All Properties property is visible and set to true, the **Tokens** component exposes

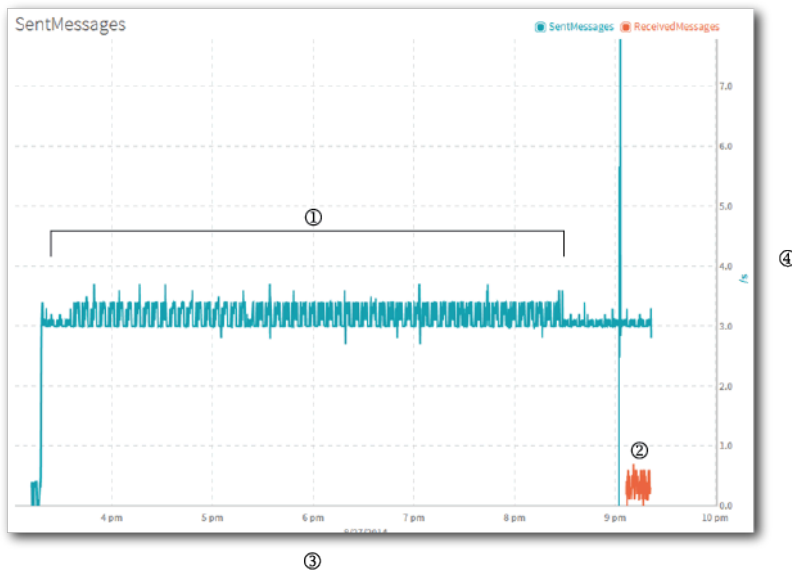
all of the current properties and any that you add in the future.

Parent topic: [Diagnosing network problems with metrics](#)

NetworkPort metrics

Counting the number of messages sent and received per second is straightforward. The SentMessages and ReceivedMessages components perform as intuitively expected.

The system monitors sent and received messages (or bytes) in the same way that it monitors tokens.



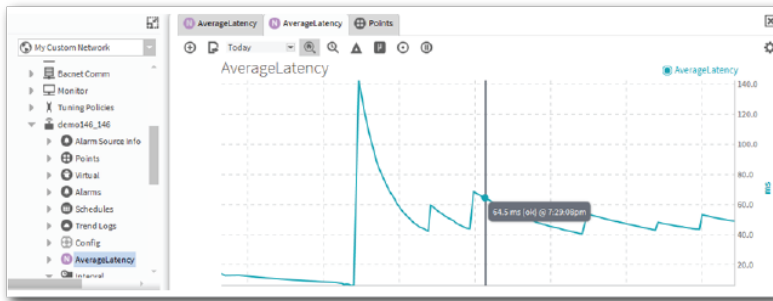
1. Sent message count
2. Received message count
3. Time
4. Per seconds

A large disparity in the number of messages sent and received can be an indicator of a larger issue. In the above example, the station is broadcasting three I-Am messages per second. This is why the received messages value is much lower and is likely to be just readProperty responses to idle device ping messages.

Parent topic: [Diagnosing network problems with metrics](#)

Device metrics

Knowing the latency of a device can be quite useful when diagnosing individual network devices that may be responding slowly, but not slowly enough to trigger an APDU timeout (APDU - Application layer Data Unit).



Tracking latency over time allows more detailed analysis of changing network conditions and can help identify problem network segments before the network or router fails. The latency of a device can also be helpful to appropriately tune the poll rate by device, instead of by network port.

Parent topic: [Diagnosing network problems with metrics](#)

Setting up a wiretap

Wiretaps evaluate (sniff) messages after they have been processed by a network port. You add wiretaps as children of the **NetworkPorts** container in the same way that you add metric components.

Your BACnet network has been set up and all devices are configured and communicating.

1. To prepare the Nav tree, expand **Config > Drivers > BacnetNetwork > Bacnet Comm > Network**.
2. To set up an MS/TP port, open the **bacnet** (Niagara BACnet Driver) palette and expand **NetworkPorts**.
3. Drag the **MstpPort** container to the **Network** container in the Nav tree.



4. To add the wiretap components, open the **bacnetUtil** palette, expand the **Wiretaps** container in the palette, and drag the **ForwardingWiretap** and **ForwardedMessageSink** to the **MstpPort** node in the Nav tree.
- [Processing a forwarded wiretap](#)
The **ForwardingWiretap** component sends each captured message to a specified IP address. The intent is to provide a way to forward MS/TP packets from a controller to another IP device, allowing the use of a packet dissection tool (for example, Wireshark) to investigate the network problem.
 - [Setting up listening for incoming packets](#)

The best solution for eliminating the ICMP reject messages is to use the ForwardedMessageSink component to listen for incoming packets. This component allows the packets to be unicast to your computer eliminating the ICMP reject messages. (ICMP — Internet Control Message Protocol). ICMP reject messages are sent back to the source device and are included in the Wireshark capture.

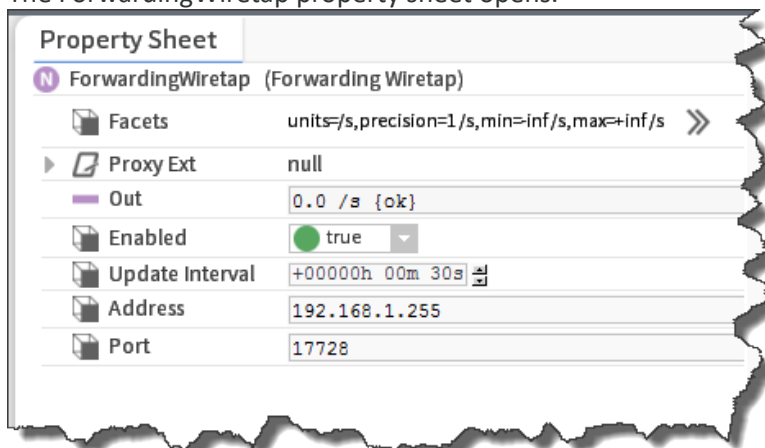
Parent topic: [BACnet troubleshooting](#)

Processing a forwarded wiretap

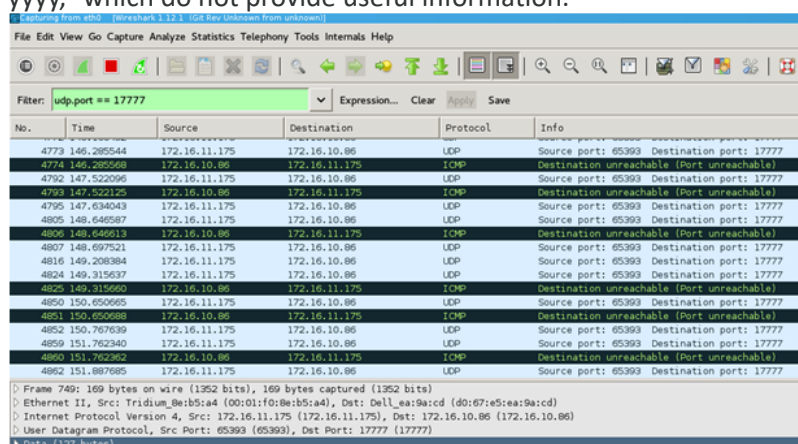
The ForwardingWiretap component sends each captured message to a specified IP address. The intent is to provide a way to forward MS/TP packets from a controller to another IP device, allowing the use of a packet dissection tool (for example, Wireshark) to investigate the network problem.

You have added the ForwardingWiretap component to the MstpPort container in the Nav tree. You are an experienced Wireshark user.

1. To configure the ForwardingWiretap component, double-click it in the Nav tree. The ForwardingWiretap property sheet opens.

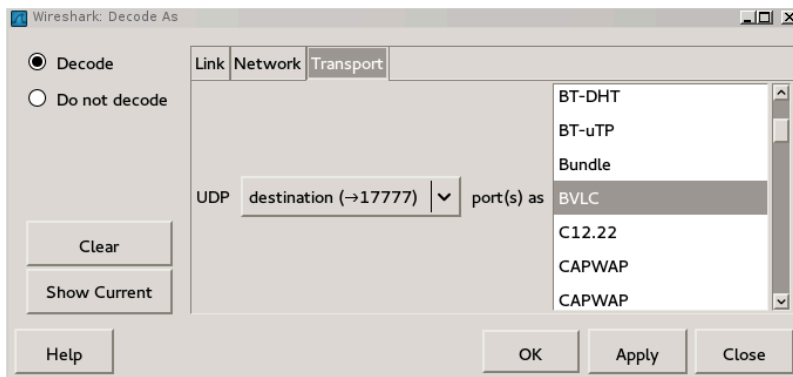


2. Change the Address field either to the IP address of your computer or to a broadcast address (if the broadcast address of the controller matches the broadcast address of the PC/laptop). Your computer should now be able to capture BACnet messages from the MS/TP trunk using Wireshark. Since the system forwards these messages on a non-standard BACnet port, which no BACnet devices are using, Wireshark needs to be configured to decode the messages as BVLC (BACnet Virtual Link Control) messages. Otherwise, the messages show up as "Source Port: xxxx" and "Destination Port: yyyy," which do not provide useful information.



3. To configure Wireshark, right-click one of the UDP (User Datagram Protocol) packets and click decode as.

A window opens for defining the destination port to associate with the protocol.



4. Select the destination you specified when you set up the ForwardWiretap properties from the **UDP** list, locate BVLC in the **port(s) as** list and click **OK**.

The system now parses the messages as BACnet-APDUs (Application Protocol Data Units):

| | | | | | | |
|------|------------|---------------|---------------|-------------|--|---------------------------|
| 5287 | 188.916698 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[199] |
| 5288 | 188.916720 | 172.16.10.86 | 172.16.11.175 | ICMP | Destination unreachable (Port unreachable) | |
| 5289 | 189.022032 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[199] |
| 5307 | 189.547874 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[200] |
| 5320 | 189.654322 | 172.16.10.86 | 172.16.11.175 | ICMP | Destination unreachable (Port unreachable) | |
| 5313 | 189.682839 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[200] |
| 5320 | 190.061785 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[201] |
| 5321 | 190.169177 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[201] |
| 5324 | 190.658662 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[202] |
| 5325 | 190.658688 | 172.16.10.86 | 172.16.11.175 | ICMP | Destination unreachable (Port unreachable) | |
| 5326 | 190.698224 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[202] |
| 5342 | 191.949935 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[203] |
| 5343 | 191.949954 | 172.16.10.86 | 172.16.11.175 | ICMP | Destination unreachable (Port unreachable) | |
| 5345 | 192.048202 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[203] |
| 5355 | 192.995761 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[204] |
| 5356 | 192.995784 | 172.16.10.86 | 172.16.11.175 | ICMP | Destination unreachable (Port unreachable) | |
| 5358 | 193.110321 | 172.16.11.175 | 172.16.10.86 | BACnet-APDU | Complex-ACK | readPropertyMultiple[204] |

This table contains quite a few ICMP (Internet Control Message Protocol) reject messages. These messages are generated by the PC's TCP/IP stack. They indicate that no process is prepared to handle these messages. In other words, this is the operating system's way of letting the caller (the controller) know that there is "nobody home."

5. Do one of the following:
 - a. Ignore the messages.
 - b. Set up a BACnet filter to omit the ICMP reject messages from the capture.
 - c. Set up a process to listen for these incoming messages and discard them.
 - d. Set up the forwarder to send the messages to a broadcast address.
 Configure sending to the broadcast address with care as every device on the network will receive the messages sent by the forwarder.

CAUTION: Do not forward messages to 47808 (0xBAC0) or any other UDP port that real BACnet devices may be listening on. The messages forwarded are properly formatted and could potentially command an unintended object to an unintended value.

Stripping out the ICMP messages leaves only the BACnet messages from the MS/TP trunk:

| | | | | | | |
|----|----------|---------------|---------------|-------------|---------------|---------------------------|
| 9 | 0.092376 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[89] |
| 11 | 1.102120 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[90] |
| 12 | 1.180433 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Complex-ACK | readPropertyMultiple[90] |
| 16 | 1.722881 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[91] |
| 18 | 1.814545 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Complex-ACK | readPropertyMultiple[91] |
| 25 | 3.111151 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Confirmed-REQ | readPropertyMultiple[92] |
| 26 | 3.208697 | 172.16.11.175 | 172.16.11.255 | BACnet-APDU | Complex-ACK | readPropertyMultiple[92] |

Parent topic: [Setting up a wiretap](#)

Setting up listening for incoming packets

The best solution for eliminating the ICMP reject messages is to use the ForwardedMessageSink component to listen for incoming packets. This component allows the packets to be unicast to your computer eliminating the ICMP reject messages. (ICMP — Internet Control Message Protocol). ICMP reject messages are sent back to the

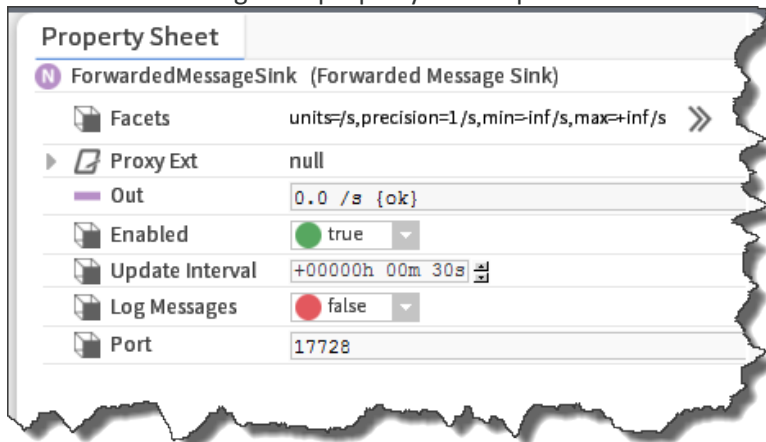
source device and are included in the Wireshark capture.

You have added the ForwardedMessageSink component to the MstpPort container in the Nav tree.

You can configure the operating system to expect these messages by setting up a station on the machine running Wireshark and directing the incoming messages to the “Forwarded Message Sink” listening on the indicated port.

Note: You do not need to run a station on your computer. You can add the ForwardedMessageSink component to any location in the station.

1. To configure the port, double-click the ForwardMessageSink. The ForwardMessageSink property sheet opens.



2. Set the Port to the same value used by the forwarder on the controller.
3. Confirm that Log Messages is false.
While setting Log Messages to true creates hex dumps of received messages in the station output, the output can be noisy. It is best to leave the property set to false and focus on the information provided by the Wireshark capture.

Parent topic: [Setting up a wiretap](#)

Overriding a device

Occasionally, BACnet devices report false capabilities or external forces make operating a device with the provided values impossible. Niagara provides override components that you can add as children of a BACnetDevice. These components provide a persistent mechanism to ignore device-provided values.

While you can temporarily alter certain properties, there is no persistent mechanism to ignore certain values from a device. In other cases, the APDU size of the device is externally influenced by an intermediary router. The device override components are:

- RpmOverride
- ApduSizeOverride
- SegmentationOverride
- ServicesOverride

Device overrides provide temporary solutions to problems that need to be corrected in third-party devices either with firmware updates or by reconfiguring the network to eliminate hourglass performance bottlenecks. As

features of last resort, these overrides should not be considered as permanent solutions. In all cases, you should notify the device manufacturer so that the root cause(s) of the problem can be addressed.

Parent topic: [BACnet troubleshooting](#)

Reference

The topics that follow provide detailed documentation for each component, plugin (view) and window that supports this system feature.

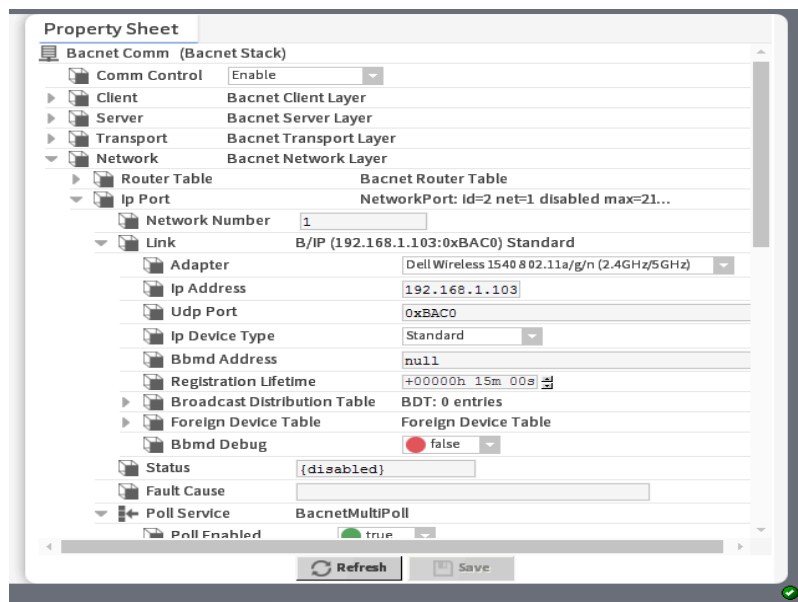
- [Network](#)
The **Network** container under **BacnetComm** determines the BACnet network-layer configuration for the station. You access **BacnetComm** directly in the Nav tree.
- [Components](#)
Components include services, folders and other model building blocks. They may be dragged and dropped onto a property or wire sheet from a palette.

Parent topic: [BACnet and Niagara 4](#)

Network

The **Network** container under **BacnetComm** determines the BACnet network-layer configuration for the station. You access **BacnetComm** directly in the Nav tree.

Figure 1. BACnet network properties



| Property | Value | Description |
|--------------------------|-------------------------|---|
| Router Table | | |
| Ip Port | | See Network properties, Ip Port . |
| Routing Enabled | true (default) or false | |
| Maintain Routing Enabled | true or false (default) | |

| Property | Value | Description |
|----------------------------|------------------------------|-------------|
| Minimum router Update time | milliseconds (default: 500) | |
| Router Discovery Timeout | milliseconds (default: 5000) | |
| Termination Time Value | seconds (default: 120) | |
| MstpPort | | |

Network properties, Ip Port

| Property | Value | Description |
|------------------------------------|-------------------------------------|---|
| Network Number | number (defaults to 1) | |
| Link | container for additional properties | B/IP (none:0xBAC0) Standard |
| Link, Adapter | drop-down list | |
| Link, Ip Address | text (defaults to none) | |
| Link, Udp Port | 0xBAC0 | Bbmd Address |
| Link, Ip Device Type | drop-down list | |
| Link, Bbmd Address | null | |
| Link, Registration Lifetime | <i>hours minutes seconds</i> | |
| Link, Broadcast Distribution Table | BDT: 0 entries | |
| Link, Foreign Device Table | | |
| Link, Bbmd Debug | true or false (default) | |
| Status | read-only | Reports the condition of the entity or process at last polling. {ok} indicates that the component is licensed and polling successfully. {down} indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. {disabled} indicates that the Enable property is set to false. {fault} indicates another problem. Refer to Fault Cause for more information. |
| Fault Cause | read-only | Indicates the reason why a system object (network, device, component, extension, etc.) is not working (in fault). This property is empty unless a fault exists. |
| Poll Service | additional properties | Configures the frequency with which the driver polls points and devices. "Poll Service properties" in the <i>Niagara Drivers Guide</i> documents these properties. |

| Property | Value | Description |
|-------------|---------------|--|
| Max Devices | max | |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Port Id | read-only | Reports the number of the Ethernet port you are configuring. |
| Port Info | read-only | Reports the type of port (Ethernet, MS/TP, etc.). |

Network properties, Ip Port, Poll Service

| Property | Value | Description |
|-------------------|------------------------------|-------------|
| Poll enabled | true (default) or false | |
| Fast Rate | <i>hours minutes seconds</i> | |
| Normal Rate | <i>hours minutes seconds</i> | |
| Slow Rate | <i>hours minutes seconds</i> | |
| Statistics Start | null | |
| Average Poll | | |
| Busy Time | | |
| Total Polls | | |
| Dibs Polls | | |
| Fast Polls | | |
| Normal Polls | | |
| Slow Polls | | |
| Dibs count | | |
| Fast Count | | |
| Normal count | | |
| Slow Count | | |
| Fast Cycle Time | | |
| Normal Cycle Time | | |
| Slow Cycle Time | | |

Network properties, MstpPort

| Property | Value | Description |
|-------------------------------|-------------------------------------|---|
| Network Number | number (default: -1) | |
| Link | MAC 0 on COM1 at Baud_9600 | |
| Link, Port Name | COM1 | |
| Link, Baud Rate | drop-down list (default: Baud_9600) | |
| Link, Mstp Address | 0-127 (default: 0) | |
| Link, Max Master | 0-127 (default: 127) | |
| Link, Max Info Frames | 0-100 (default: 20) | |
| Link, Support Extended Frames | true or false (default) | |
| Status | read-only | Reports the condition of the entity or process at last polling. {ok} indicates that the component is licensed and polling successfully. {down} indicates that the last check was unsuccessful, perhaps because of an incorrect property, or possibly loss of network connection. {disabled} indicates that the Enable property is set to false. {fault} indicates another problem. Refer to Fault Cause for more information. |
| Fault Cause | read-only | Indicates the reason why a system object (network, device, component, extension, etc.) is not working (in fault). This property is empty unless a fault exists. |
| Poll Service | | Configures the frequency with which the driver polls points and devices. "Poll Service properties" in the <i>Niagara Drivers Guide</i> documents these properties. |
| Max Devices | max | |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Port Id | read-only | Reports the number of the Ethernet port you are configuring. |
| Port Info | read-only | Reports the type of port (Ethernet, MS/TP, etc.). |

Parent topic: [Reference](#)

Components

Components include services, folders and other model building blocks. They may be dragged and dropped onto a property or wire sheet from a palette.

The descriptions included in the following topics appear as headings in documentation. They also appear as context-sensitive help topics when accessed by:

- Right-clicking on the component and selecting **Views > Guide Help**
- Clicking **Help > Guide On Target**.
- [ApduSizeOverride](#)
This component allows you to specify a custom APDU (Application Protocol Data Units) size for a device.
- [AverageLatency](#)
You can add this component to a BBacnetDevice for the purpose of recording the average amount of time it takes the device to respond to a ping or poll message.
- [ForwardingWiretap](#)
This component sends each captured message to a specified IP address. Once the message arrives, you may use a packet dissection tool, such as Wireshark, to investigate network problems.
- [ForwardedMessageSink](#)
This component listens for incoming packets and unicasts them to your computer eliminating the ICMP reject messages that clutter the output from a ForwardingWiretap.
- [RpmOverride](#)
You can use this component to ignore the reported value for RPM (readPropertyMultiple).
- [SegmentationOverride](#)
This component causes the system to persistently ignore the segmentation support of the BBacnetDevice.
- [SentMessages](#)
These components capture the messages exchanged between a controller and client. The properties are the same for both sent and received message components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.
- [ReceivedMessages](#)
These components capture the messages exchanged between a controller and client. The properties are the same for both sent and received message components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.
- [SentBytes](#)
These components capture the individual bytes exchanged between a controller and client. The properties are the same for both sent and received byte components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.
- [Received Bytes](#)
These components capture the individual bytes exchanged between a controller and client. The properties are the same for both sent and received byte components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.
- [ServicesOverride](#)
When added to a BBacnetDevice, this component causes the system to ignore the claimed services supported.
- [Tokens](#)
This component is a BNumericPoint that can be tracked, alarmed, and charted like any other NumericPoint. Using it improves the ability to diagnose certain configuration problems. While the problems may require a site visit to resolve, the metrics may provide a valuable diagnostic starting point.

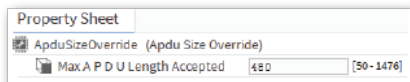
Parent topic: [Reference](#)

ApduSizeOverride

This component allows you to specify a custom APDU (Application Protocol Data Units) size for a device.

Being able to customize the APDU size helps with a router that has a lower APDU size in between two nodes that claim a full-size APDU (for example 1476). Without the ability to reject messages that are too large, the intermediate router silently drops the oversized RPM messages. Adding an ApduSizeOverride to a device ignores the device-provided value and allows the system to work around hourglass networks.

Figure 1. ApduSizeOverride property



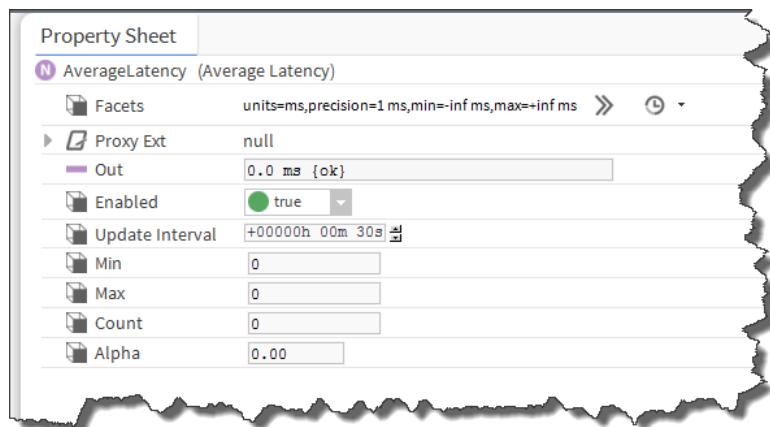
| Property | Value | Description |
|-----------------------------|------------------------|-----------------------|
| Max A P D U Length Accepted | number from 50 to 1476 | The size of the APDU. |

Parent topic: [Components](#)

AverageLatency

You can add this component to a BBacnetDevice for the purpose of recording the average amount of time it takes the device to respond to a ping or poll message.

Figure 1. Average Latency properties



| Property | Value | Description |
|----------|------------------------|--|
| Facets | depends on the context | Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types. With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value. Besides control points, various other |

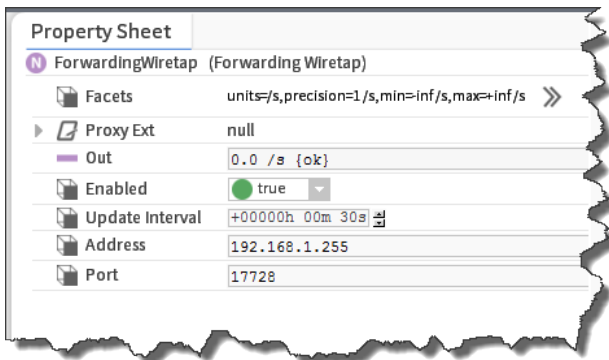
| Property | Value | Description |
|-----------------|------------------------------|--|
| | | <p>components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out | read-only | <p>Displays the current value of the proxy point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Update Interval | | |
| Min | | |
| Max | | |
| Count | | |
| Alpha | | |

Parent topic: [Components](#)

ForwardingWiretap

This component sends each captured message to a specified IP address. Once the message arrives, you may use a packet dissection tool, such as Wireshark, to investigate network problems.

Figure 1. ForwardingWiretap properties



| Property | Value | Description |
|---------------------|------------------------------|--|
| Facets | depends on the context | <p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | <p>Displays the current value of the proxy point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by</p> |

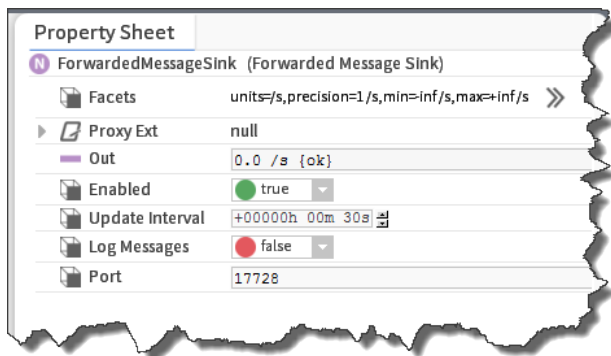
| Property | Value | Description |
|-----------------|---------------|--|
| | | a combination of status flags, such as <code>fault</code> , <code>overridden</code> , <code>alarm</code> , and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code> . |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Update Interval | read-only | Reports the frequency of updates. |
| Ip Address | IP address | Identifies a device, which is connected to a network that uses the Internet Protocol for communication. |
| Port Number | number | Defines the port number on the controller or computer used to connect to the network. If using fox streaming, which uses the station to render the video stream, this port should be different from the station's fox port. If you are not using fox streaming, this port should be the same as the station's fox port. |

Parent topic: [Components](#)

ForwardedMessageSink

This component listens for incoming packets and unicasts them to your computer eliminating the ICMP reject messages that clutter the output from a ForwardingWiretap.

Figure 1. ForwardedMessageSink properties



| Property | Value | Description |
|----------|------------------------|--|
| Facets | depends on the context | Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types. |

| Property | Value | Description |
|---------------------|------------------------------|--|
| | | <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | <p>Displays the current value of the proxy point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |
| Enabled | true or false | Activates (true) and deactivates (false) use of the object (network, device, point, component, table, schedule, descriptor, etc.). |
| Update Interval | read-only | Reports the frequency of updates. |
| Log Messages | true or false | When set to true, the system creates a hex dump of each received message and stores it in station output. Setting this property to false disables the creation of a hex dump of each received message. |
| Port Number | number | Defines the port number on the controller or computer used to connect to the network. |

| Property | Value | Description |
|----------|-------|---|
| | | If using fox streaming, which uses the station to render the video stream, this port should be different from the station's fox port. If you are not using fox streaming, this port should be the same as the station's fox port. |

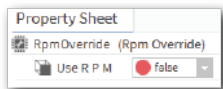
Parent topic: [Components](#)

RpmOverride

You can use this component to ignore the reported value for RPM (readPropertyMultiple).

By default Niagara's poll service uses a bin packing algorithm to fit as many requests into a readPropertyMultiple request as the device claims it can fit into a response. Some devices may not be able to process a maximized readPropertyMultiple message faster than the APDU timeout. The RpmOverride provides a way to ignore the device's support of RPM. The poll service continues to poll the device using single readProperty messages for each subscribed property.

Figure 1. RpmOverride property



| Property | Value | Description |
|-----------|---------------|--|
| Use R P M | true or false | <p>true sends RPM messages to the device. This is the case even if the device does not support these messages. Expect many unsupported service errors to be returned by the device.</p> <p>false causes the system to ignore any claimed support for RPM. The system only sends readProperty messages to the device, even if the device supports readPropertyMultiple.</p> |

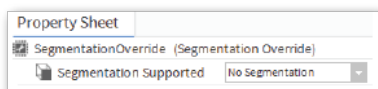
Parent topic: [Components](#)

SegmentationOverride

This component causes the system to persistently ignore the segmentation support of the BBacnetDevice.

Some devices claim segmentation support but either do not actually support segmentation or support it poorly. Other devices consistently send segments out of order. Unsegmented requests may experience better sustained throughput than large segmented messages.

Figure 1. SegmentationOverride property



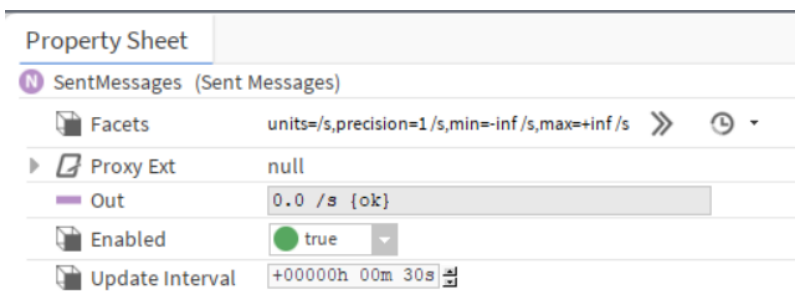
| Property | Value | Description |
|------------------------|----------------|-------------|
| Segmentation Supported | drop-down list | |

Parent topic: [Components](#)

SentMessages

These components capture the messages exchanged between a controller and client. The properties are the same for both sent and received message components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.

Figure 1. SentMessages



| Property | Value | Description |
|---------------------|------------------------------|--|
| Facets | depends on the context | <p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | Displays the current value of the proxy point including facets and status. |

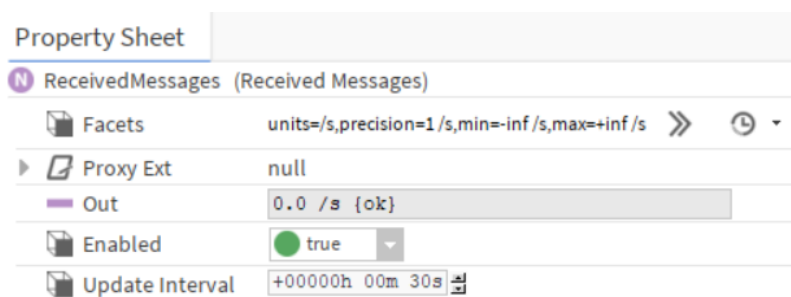
| Property | Value | Description |
|-----------------|---------------|--|
| | | <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |
| Enabled | true or false | Indicates if the network, device, point or component is active or inactive. |
| Update Interval | read-only | Reports the frequency of updates. |

Parent topic: [Components](#)

ReceivedMessages

These components capture the messages exchanged between a controller and client. The properties are the same for both sent and received message components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.

Figure 1. ReceivedMessages



| Property | Value | Description |
|----------|------------------------|---|
| Facets | depends on the context | <p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other</p> |

| Property | Value | Description |
|---------------------|------------------------------|--|
| | | <p>components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | <p>Displays the current value of the proxy point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |
| Enabled | true or false | Indicates if the network, device, point or component is active or inactive. |
| Update Interval | read-only | Reports the frequency of updates. |

Parent topic: [Components](#)

SentBytes

These components capture the individual bytes exchanged between a controller and client. The properties are the same for both sent and received byte components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.

Figure 1. SentBytes

Property Sheet

N SentBytes (Sent Bytes)

Facets units=B/s,precision=1 B/s,min=-inf B/s,max=+inf B/... >> ⌚ ▾

Proxy Ext null

Out 0.0 B/s {ok}

Enabled ☒ true ▾

Update Interval +000000h 00m 30s ⌚

| Property | Value | Description |
|---------------------|------------------------------|--|
| Facets | depends on the context | <p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | <p>Displays the current value of the proxy point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |

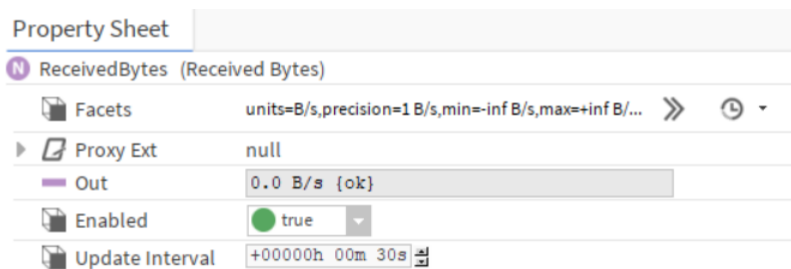
| Property | Value | Description |
|-----------------|---------------|---|
| Enabled | true or false | Indicates if the network, device, point or component is active or inactive. |
| Update Interval | read-only | Reports the frequency of updates. |

Parent topic: [Components](#)

Received Bytes

These components capture the individual bytes exchanged between a controller and client. The properties are the same for both sent and received byte components. Once set up, you can view a chart that summarizes the information captured by right-clicking on the component and clicking **Views > Chart**.

Figure 1. Received Bytes



| Property | Value | Description |
|---------------------|------------------------------|--|
| Facets | depends on the context | <p>Determine how values are formatted for display depending on the context and the type of data. Examples include engineering units and decimal precision for numeric types, and descriptive value (state) text for boolean and enum types.</p> <p>With the exception of proxy points (with possible defined device facets), point facets do not affect how the framework processes the point's value.</p> <p>Besides control points, various other components have facets too. For example, many kitControl and schedule components have facets. Details about point facets apply to these components too, unless especially noted.</p> <p>You access facets by clicking an Edit button or a chevron >>. Both open an Edit Facets window.</p> |
| Proxy ext | Out, In, and Fallback values | <p>Indicate from where the point's value originates, including details specific to the parentage of the point's network and communications (driver).</p> <p><code>null</code> indicates that the point is an empty placeholder.</p> |
| BACnet Out Property | read-only | Displays the current value of the proxy |

| Property | Value | Description |
|-----------------|---------------|--|
| | | <p>point including facets and status.</p> <p>The value depends on the type of control point.</p> <p>Facets define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. You can edit point facets to poll for additional properties, such as the native statusFlags and/or priorityArray level.</p> <p>Status reports the current health and validity of the value. Status is specified by a combination of status flags, such as <code>fault</code>, <code>overridden</code>, <code>alarm</code>, and so on. If no status flag is set, status is considered normal and reports <code>{ok}</code>.</p> |
| Enabled | true or false | Indicates if the network, device, point or component is active or inactive. |
| Update Interval | read-only | Reports the frequency of updates. |

Parent topic: [Components](#)

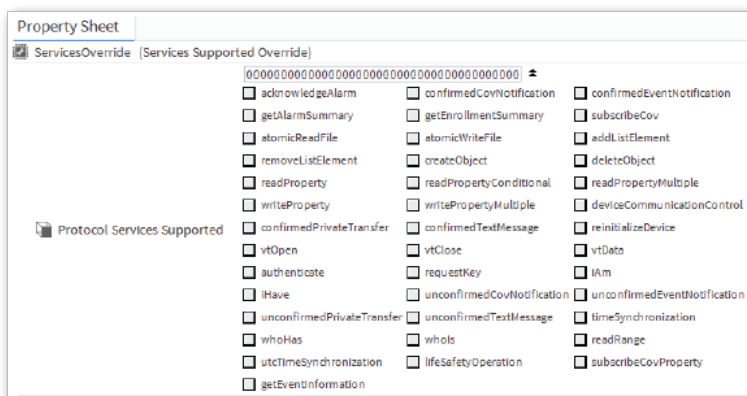
ServicesOverride

When added to a B BacnetDevice, this component causes the system to ignore the claimed services supported.

While the RpmOverride may be sufficient as it is the service that users have requested to override, there may be some other service you wish to ignore.

By default, the component supports no services until the **Reset** action is performed on the this component.

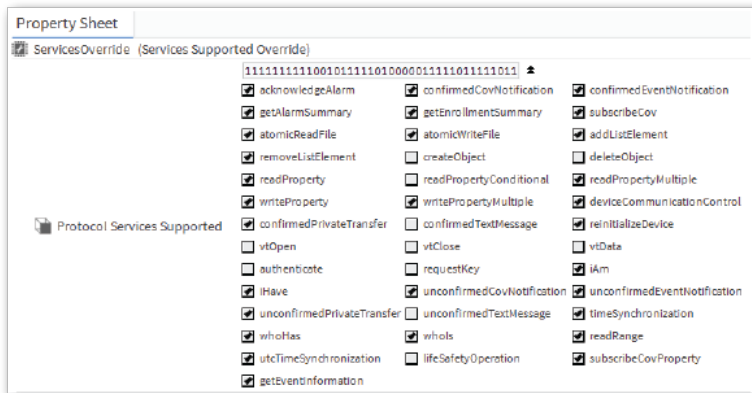
Figure 1. ServicesOverride properties



As a child of a B BacnetDevice, the ServicesOverride component loads the claimed services from the target device using the **reset** action. As an independent object, the **reset** action clears the services supported.

For example, after running the reset action:

Figure 2. ServicesOverride example

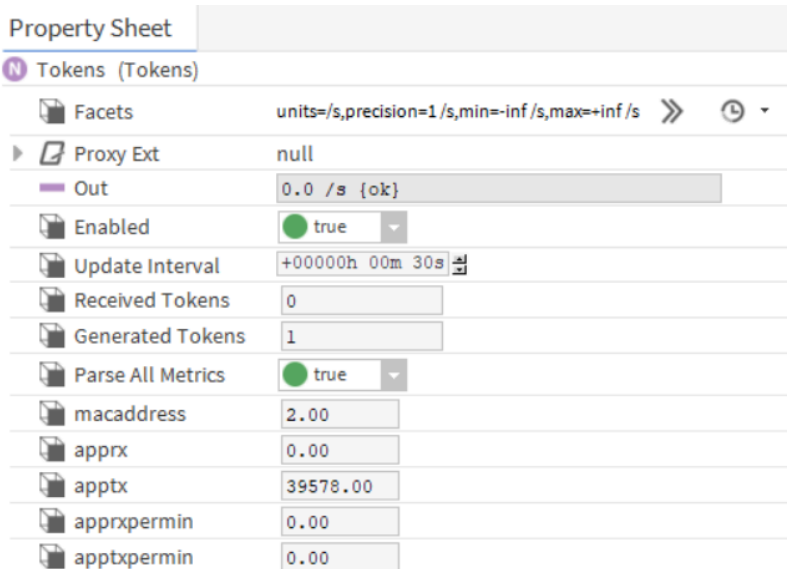


Once you select or clear each protocol and save, Niagara uses the overrides in preference to device-provided values.

Parent topic: [Components](#)

Tokens

This component is a BNumericPoint that can be tracked, alarmed, and charted like any other NumericPoint. Using it improves the ability to diagnose certain configuration problems. While the problems may require a site visit to resolve, the metrics may provide a valuable diagnostic starting point.



| Type | Category/Value | Legacy Mode (Tridium485-2_r06 and earlier) | Coprocessor Mode (Tridium485-2_r09 and later) | Description |
|------------------|-----------------------|--|--|--------------|
| GloBAC_rx_brdcst | incrementing count | x | | Experimental |

| | | | | |
|-------------------|--|---|---|--|
| 6loBAC_tx_brdcst | incrementing count | x | | Experimental |
| apprx | incrementing count | x | x | Number of frames containing BACnet data to this node, or broadcast that have been received (EXPECTING_REPLY and NOT_EXPECTING_REPLY) |
| aprxpermin | rate | x | x | Number of BACnet data frames received per minute to this node's address (or broadcast) |
| apptx | incrementing count | x | x | Number of frames containing BACnet data transmitted from this node |
| apptxpermin | rate | x | x | Number of BACnet data frames per minute transmitted from this node (directed and broadcast) |
| apptxq_full | incrementing count | | x | The number of times that both positions of the 2-deep app tx queue are full in the SAM4S. This is a normal condition. |
| apptxq_invalid | incrementing count | | x | The number of times that the 2-deep app tx queue is in an invalid state, for example, both buffers are in the "waiting to transmit" state or an undefined state. This should never happen. |
| apptxq_overflow | incrementing count | | x | The number of times that both positions of the 2-deep app tx queue are full and we attempt to add another message from the host. This is an abnormal condition and should not happen under normal circumstances. |
| baddatacrc16 | incrementing count | x | x | The number of bad data CRC16s (FT_BACNET_DATA_EXPECTING_REPLY and FT_BACNET_DATA_NOT_EXPECTING_REPLY) |
| baddatacrc32 | incrementing count | x | x | The number of bad data CRC32s (FT_BACNET_EXT_DATA_EXPECTING_REPLY and FT_BACNET_EXT_DATA_NOT_EXPECTING_REPLY) |
| badheadercrc | incrementing count | x | x | The number of bad header CRCs |
| baudrate | link layer config | x | x | 9600 19200 38400 57600 76800 115200, default 115200 |
| char time ns | calculated from baud rate/ nanoseconds per byte | x | | Used in 40 bit time and 60 bit time calculations |
| cycle sleep | command line arg/defaults to 40, in 1/10ms | x | | Adjusts time that mstp daemon sleeps when in idle state. Set by "-a" arg to mstp daemon |
| declaresolemaster | incrementing count | x | x | Incremented by MNSM, the number of times this node has entered the "sole master" |
| emstpivalcmd | incrementing count | | x | Received a message from the host that is not a request, or an unrecognized command type. All |

| | | | | |
|------------------|--------------------------------|---|---|---|
| | | | | messages from the host should be a emstp request. |
| emstpinvallen | incrementing count | | x | Number of bytes from the host does not match the size decoded from the emstp header. |
| emstpinvalproto | incrementing count | | x | In coprocessor mode, if the first byte of a message from the host does not start with 0x01, this count is incremented and the bytes are thrown away. |
| emstpkeepaliveto | incrementing count | | x | Emstp “keep alives” are sent from SAM4S to host if 30 seconds have expired with no emstp messages from the host. |
| emstprxqsize | Host side metric, snapshot | | x | The number of messages received from the coprocessor and awaiting processing by the link layer. This queue is 5 messages deep. |
| emstptxqsize | Host side metric, snapshot | | x | The number of messages in the transmitQueue at any given time. This queue 32 messages deep. |
| extrxpermin | rate | x | x | The messages/minute of FT_BACNET_EXT_DATA_EXPECTING_REPLY or FT_BACNET_EXT_DATA_NOT_EXPECTING_REPLY received |
| exttxpermin | rate | x | x | The messages/minute of FT_BACNET_EXT_DATA_EXPECTING_REPLY or FT_BACNET_EXT_DATA_NOT_EXPECTING_REPLY minute |
| extframes_rx | incrementing count | x | x | The number of FT_BACNET_EXT_DATA_EXPECTING_REPLY or FT_BACNET_EXT_DATA_NOT_EXPECTING_REPLY received |
| extframes_tx | incrementing count | x | x | The number of FT_BACNET_EXT_DATA_EXPECTING_REPLY or FT_BACNET_EXT_DATA_NOT_EXPECTING_REPLY transmitted |
| framecount | MNSM snapshot | x | x | Range: 0 to maxInfoFrames. The number of frames sent by this node during the current token hold. When reaching "max info frames", the node must pass the token. |
| generatedTokens | incrementing count | x | x | Incremented by MNSM, the number of tokens generated by this node |
| invalidframes | incrementing count | x | x | Incremented by RFSM, the total number of frames that could not be properly decoded. Included incomplete frames, frames with bad header or data CRC, frames with invalid length in the header. |
| ioerrorgetavail | Host side metric, incrementing | | x | The call to get the number of bytes available from the serial input buffer returning an error |

| count | | | | |
|--------------------------------|---|---|---|---|
| ip_6loBAC_rx | incrementing count | x | | Experimental |
| ip_6loBAC_tx | incrementing count | x | | Experimental |
| keepalivesent | Host side metric, incrementing count | | x | The number of times the host has sent a “keep alive”. A keep alive is sent if 5 seconds of elapsed time with no other messages to send. This can happen, for example, on station start-up, long GC cycles, station save when there are not enough CPU cycles for the link layer poll threads. |
| macaddress | Link layer config/ ranges from 0-127, defaults to 0 | x | x | |
| master_state | MNSM snapshot | x | x | 0=init, 1=idle, 2=use token, 3=wait for reply, 4=done with token, 5=pass token, 6=no token, 7=poll for master, 8=answer data request |
| maxinfoframes | Link layer config/ ranges from 1-127, defaults to 50 | x | x | |
| maxmaster | Link layer config/ ranges from 0-127, defaults to 127 | x | x | |
| n40bitdelay (coprocessor mode) | Calculated from baud rate/values 2 3 4 6 ms | | x | tTurnaround, baud rate dependant: 6ms@9600, 4ms@19200, 3ms@38400, 2ms@57600 and above |
| n40bitdelay (legacy mode) | Calculated from baud rate/ nanoseconds for 40 bit delay (40 * 1 billion / baud) | x | | |
| nextstation | MNSM snapshot | x | x | The MAC address of the node to which this station passes the token. If unknown, Next_Station = this station's MAC address. |
| pollstation | MNSM snapshot | x | x | The MAC address of the node to which this station last sent a PFM. |
| receive_state | RFSM snapshot | x | x | 0=idle, 1=preamble, 2=header, 3=data, 4=skip data, 5=cobs data, 6=cobs crc |
| receivedTokens | incrementing count | x | x | Incremented by MNSM, the number of tokens received while in idle state |
| retrytokencnt | MNSM snapshot/values 0 or 1 | x | x | MS/TP is allowed to resend the FT_TOKEN up to a maximum of 2 total if the first times out after tUsage timeout. |

| | | | | |
|---------------------------------|-----------------------------|---|---|---|
| ringbufoverflow | incrementing count | | x | Number of times a received byte was dropped because the 64-byte serial ring buffer (serial rx buffer) was full |
| rxbytespersec | incrementing count | x | x | The bytes per second received over the last 10 seconds |
| rxbytestotal | incrementing count | x | x | The total number of bytes received from the RS485 line |
| rxQ queued msgs | MNSM snapshot | x | | The number of application messages in the rxq awaiting processing by BACnet driver |
| rxQ_full count | incrementing count | x | | The number of times the MNSM could not put a message into the rxQ, to be processed by upper BACnet layers because the queue was full |
| serialport | Link layer config | x | x | COM1, COM2, etc. |
| serialTxFailures | incrementing count | x | | The number of times a write() to the serial output stream failed |
| serrxframingerr | incrementing count | | x | Number of UART character framing errors reported |
| silencetimer (coprocessor mode) | MNSM snapshot | | x | SAM4S coprocessor mode: incremented every millisecond when there is no tx or rx character on the serial bus, reset on tx or rx of byte. Used to measure and generate silence on the serial bus. |
| silencetimer (legacy mode) | MNSM snapshot | x | | Legacy mode: elapsed ms time calculated from monotonic clock every RFSM and/or MNSM execution, reset on tx or rx of a byte. Used to measure and generate silence on the serial bus. |
| skippedframes | incrementing count | x | x | The number of data containing frames that are not addressed to this node and are consumed by the RFSM SKIP_DATA state |
| solemaster | MNSM snapshot/range: 0 or 1 | x | x | A boolean flag set to 1 by the MNSM if this node is the only known master node |
| t_usage_0_5ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 5ms |
| t_usage_5_10ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 5-10ms |
| t_usage_10_15ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 10-15ms |
| t_usage_15_20ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 15-20ms |
| t_usage_20_35ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 20-35ms |
| t_usage_35_85ms | incrementing count | x | x | The number of times a remote node responded to a token pass within 35-85ms |

| | | | | |
|------------------|--|---|---|---|
| t_usage_85_plus | incrementing count | x | x | The number of times a remote node responded to a token pass greater than 85ms |
| t_usage_err | incrementing count | x | | tUsageStart is greater than tUsageEnd, N/A for coprocessor mode |
| testrequest_rx | incrementing count | x | x | The number of FT_TEST_REQUEST frames received |
| testrequest_tx | incrementing count | x | x | The number of FT_TEST_REQUEST frames transmitted |
| testresponse_rx | incrementing count | x | x | The number of FT_TEST_RESPONSE frames received |
| testresponse_tx | incrementing count | x | x | The number of FT_TEST_RESPONSE frames transmitted |
| tokencount | MNSM snapshot/values 0-50 | x | x | The number of tokens received by this node. When 50 tokens have been received, the node must perform PFM cycle. |
| trunk | Link layer config | | x | mstp1, mstp2, etc. |
| txbytespersec | incrementing count | x | x | The bytes per second transmitted over the last 10 seconds |
| txbytestotal | incrementing count | x | x | The total number of bytes transmitted onto the RS485 line |
| txmsgbufferfull | Host side metric, incrementing count | | x | The queue which buffers NPDU messages from the link layer is 32 messages deep. This is a count of how many times this queue fills. Indicative of polling faster than messages can be processed. |
| txQ queued msgs | MNSM snapshot | x | | The number of application messages in the tx queue from the BACnet driver at time of snapshot |
| txQ_full count | incrementing count | x | | The number of times a message from BACnet layers via DCMD_MSTP_TX_FRAME, could not put a message in the tx queue because it was full |
| txthrottle | Link layer config/ range: 0-20ms, defaults to 10ms | x | x | Only comes into play when this node owns the token. Inserts a delay between tx'd frames. |
| unexpectedframes | incrementing count | x | x | The number of frames received while the MNSM state was in a state that did not expect them. This may be due to the following: 1) Unexpected frame, such as token or pfm, while waiting for a response to a frame that expects a reply 2) A broadcast frame received while waiting for a response 3) Any frame other than a RPFM while waiting for a reply to poll for master |

| | | | | |
|-------------------|---|---|---|---|
| | | | | <p>4) Any frame type other than test response while waiting for a testresponse</p> <p>5) Receiving a test response while MNSM is in idle state</p> |
| unknownEmstpMsgs | Host side metric, incrementing count | | x | An unknown protocol byte, command, or inconsistent emstp message length was received from the coprocessor. |
| unwantedframes | incrementing count | x | x | Count of unwanted frames, such as a token pass or pfm to the broadcast address, receive a reply to poll for master while in idle state, or receive a reply postponed while in idle state, or a proprietary frame type directed to this station which this station does not support. |
| usage timeout: 20 | Link layer config/ values 20 35, defaults to 20 | x | x | The time without a received byte that this node must wait after sending a token or PFM to allow the receiving node to begin replying |
| usartindex | Range: 0 or 1 | | x | The hardware index of the USART used on the coprocessor (sam4s) |
| usbtxfailures | incrementing count | | x | Count of the number of times that a message from sam4s to host did not send all bytes in one write |
| validframes | incrementing count | x | x | Incremented by RFSM, the total number of all frames properly decoded |
| version | text string | x | x | A version string embedded in the coprocessor firmware, such as "MS/TP Coprocessor 2.249, Jan 12 2022" |

Parent topic: [Components](#)