Technical Document

Engineering Notes

December 14, 2015

niagara4

Engineering Notes

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

Contents

About this guide

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

Updated, December 14, 2015. Appended usage notes for BACnet in N4 and use of **`bacnetUtil`** components.

Initial release publication, August 19, 2015.

Related documentation

Related information is available in the following documents.

- *Getting Started with Niagara*
- *Niagara 4 Platform Guide*

Chapter 1   Niagara Display Names

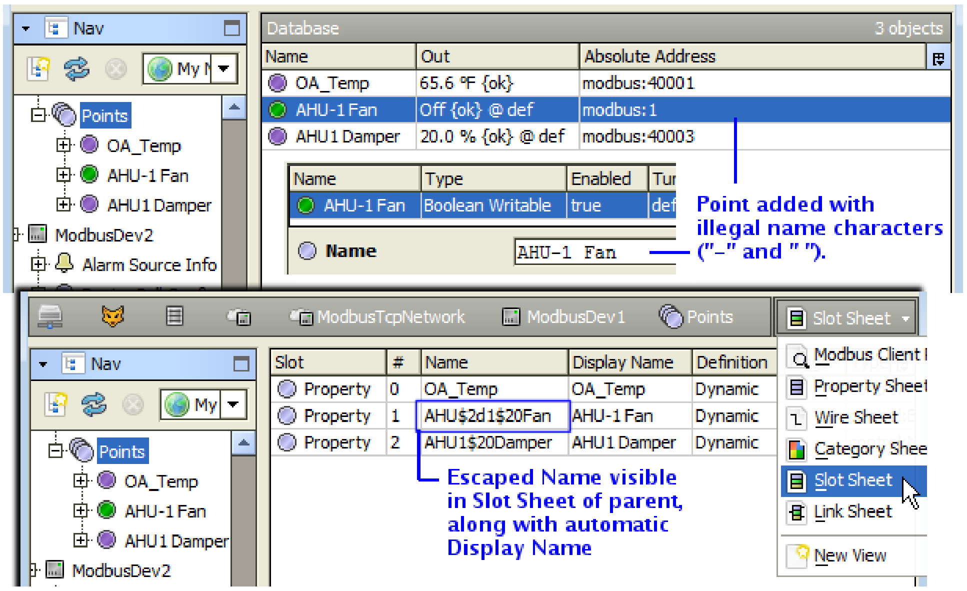
Topics covered in this chapter

♦ Automatic display names
♦ Display name storage
♦ Manually setting display names
♦ Using the Batch Editor for display names

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

Display names are an available feature on components, component slots, and histories in a station. For a component, display name differs from *name*, in that the name must always follow certain rules, and is included as a portion of the *ord* for that 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



Name	Out	Absolute Address
OA_Temp	65.6 °F {ok}	modbus:40001
AHU-1 Fan	Off {ok} @ def	modbus:1
AHU1 Damper	20.0 % {ok} @ def	modbus:40003

Name	Type	Enabled	Tur
AHU-1 Fan	Boolean Writable	true	def

Name AHU-1 Fan

Point added with illegal name characters ("-" and " ").

Slot	#	Name	Display Name	Definition
Property	0	OA_Temp	OA_Temp	Dynamic
Property	1	AHU$2d1$20Fan	AHU-1 Fan	Dynamic
Property	2	AHU1$20Damper	AHU1 Damper	Dynamic

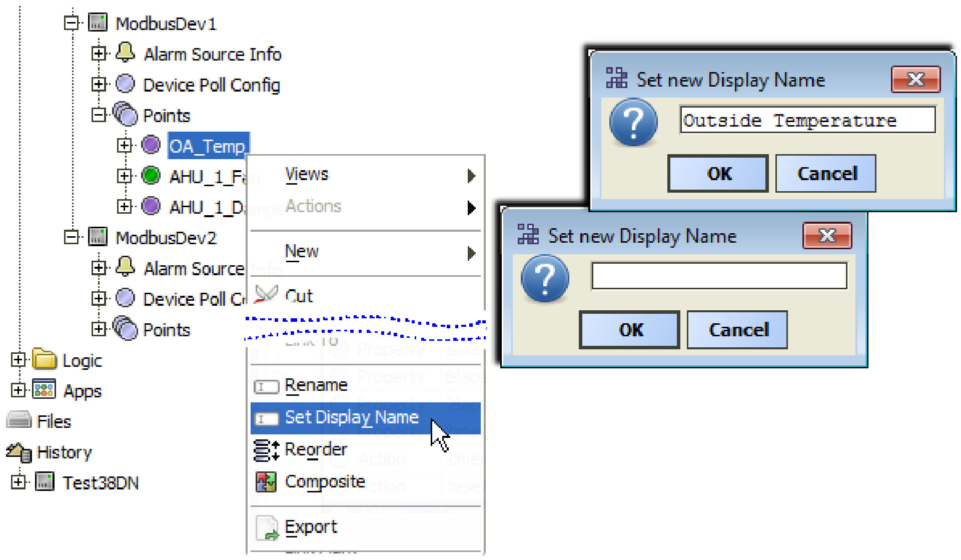
Escaped Name visible in Slot Sheet of parent, along with automatic Display Name

If you add a component with a name that breaks such 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.

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

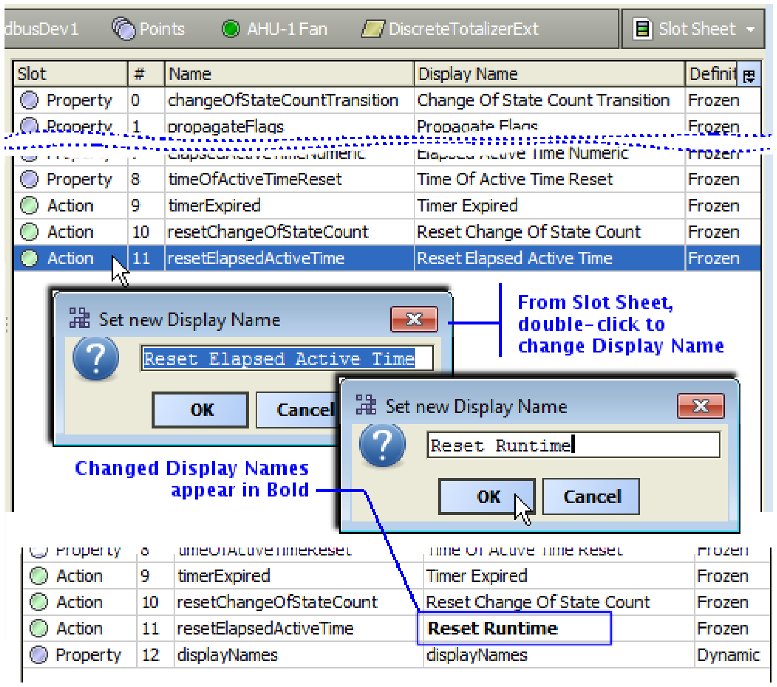


Outside Temperature

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.

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



Slot	#	Name	Display Name	Definit
Property	0	changeOfStateCountTransition	Change Of State Count Transition	Frozen
Property	1	propagateFlags	Propagate Flags	Frozen
		elapsedActiveTimeNumeric	Elapsed Active Time Numeric	Frozen
Property	8	timeOfActiveTimeReset	Time Of Active Time Reset	Frozen
Action	9	timerExpired	Timer Expired	Frozen
Action	10	resetChangeOfStateCount	Reset Change Of State Count	Frozen
Action	11	resetElapsedActiveTime	Reset Elapsed Active Time	Frozen

Reset Elapsed Active Time

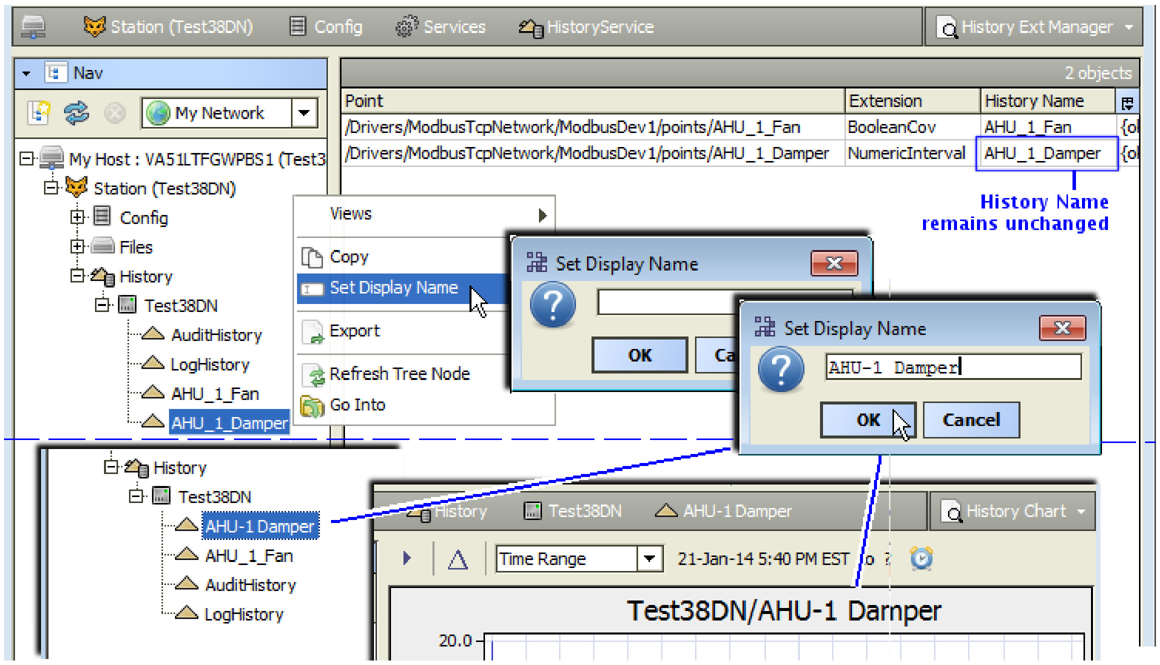
Reset Runtime

Property	8	timeOfActiveTimeReset	Time Of Active Time Reset	Frozen
Action	9	timerExpired	Timer Expired	Frozen
Action	10	resetChangeOfStateCount	Reset Change Of State Count	Frozen
Action	11	resetElapsedActiveTime	**Reset Runtime**	Frozen
Property	12	displayNames	displayNames	Dynamic

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

Starting in AX-3.6, *histories* also support display names. In the Workbench Nav tree, simply right-click a history for the **Set Display Name** command.

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



Point	Extension	History Name
/Drivers/ModbusTcpNetwork/ModbusDev1/points/AHU_1_Fan	BooleanCov	AHU_1_Fan
/Drivers/ModbusTcpNetwork/ModbusDev1/points/AHU_1_Damper	NumericInterval	AHU_1_Damper

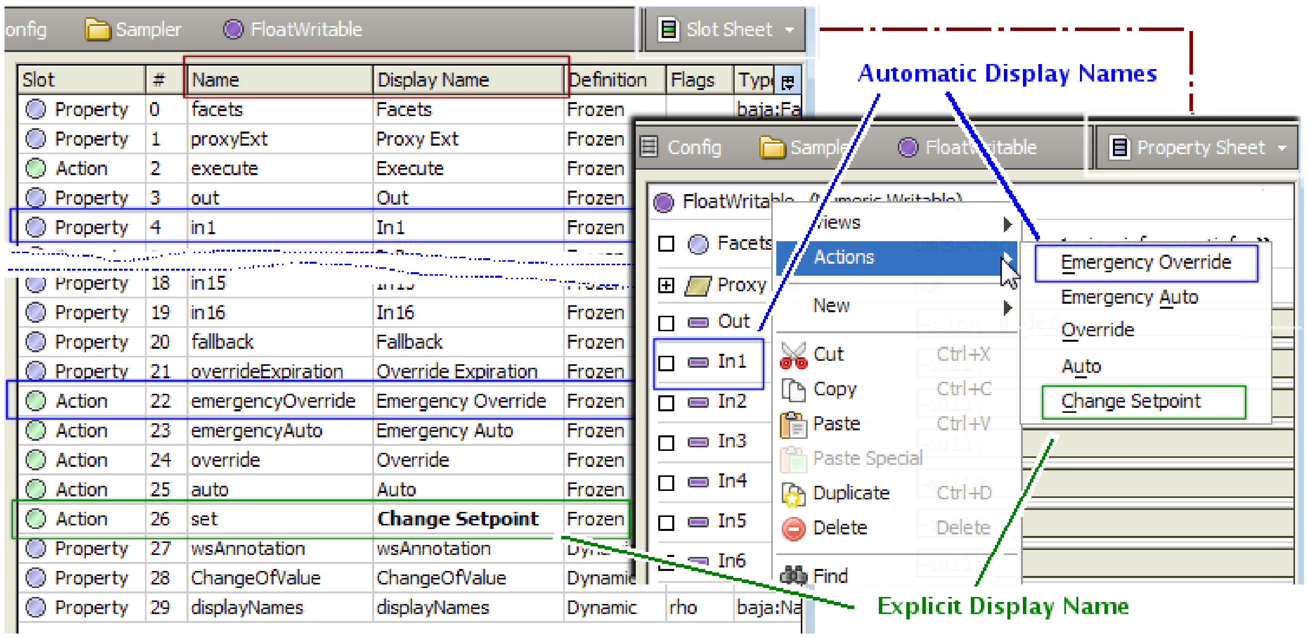
Note if you assign a display name to a history, its original history ID is not affected. The display name is simply used when displaying the history in various views into the system.

Automatic display names

Niagara 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 5 Automatic display names for frozen slots vs. explicit display names



Slot	#	Name	Display Name	Definition	Flags	Typ
Property	0	facets	Facets	Frozen		baja:Fa
Property	1	proxyExt	Proxy Ext	Frozen		
Action	2	execute	Execute	Frozen		
Property	3	out	Out	Frozen		
Property	4	in1	In1	Frozen		
Property	18	in15				
Property	19	in16	In16	Frozen		
Property	20	fallback	Fallback	Frozen		
Property	21	overrideExpiration	Override Expiration	Frozen		
Action	22	emergencyOverride	Emergency Override	Frozen		
Action	23	emergencyAuto	Emergency Auto	Frozen		
Action	24	override	Override	Frozen		
Action	25	auto	Auto	Frozen		
Action	26	set	**Change Setpoint**	Frozen		
Property	27	wsAnnotation	wsAnnotation	Dyn		
Property	28	ChangeOfValue	ChangeOfValue	Dynamic		
Property	29	displayNames	displayNames	Dynamic	rho	baja:Na

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".

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

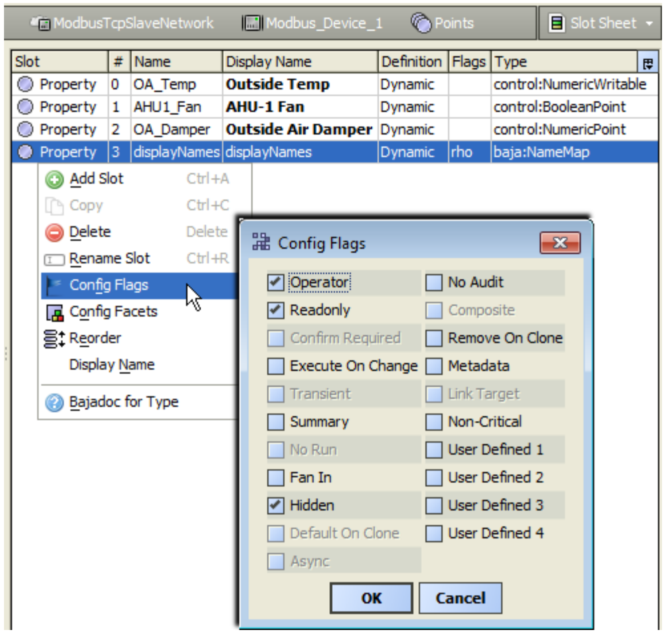
The "`displayNames`" slot on the parent component uses the data type `baja:NameMap`. This slot is created upon the first explicit display name given to a child slot. If the `displayNames` slot already exists, any new explicit display name is appended to its BNameMap value.

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 6 *Default config flags for displayNames slot of a component or container*



ModbusTcpSlaveNetwork | Modbus_Device_1 | Points | Slot Sheet

Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	OA_Temp	**Outside Temp**	Dynamic		control:NumericWritable
Property	1	AHU1_Fan	**AHU-1 Fan**	Dynamic		control:BooleanPoint
Property	2	OA_Damper	**Outside Air Damper**	Dynamic		control:NumericPoint
Property	3	displayNames	displayNames	Dynamic	rho	baja:NameMap

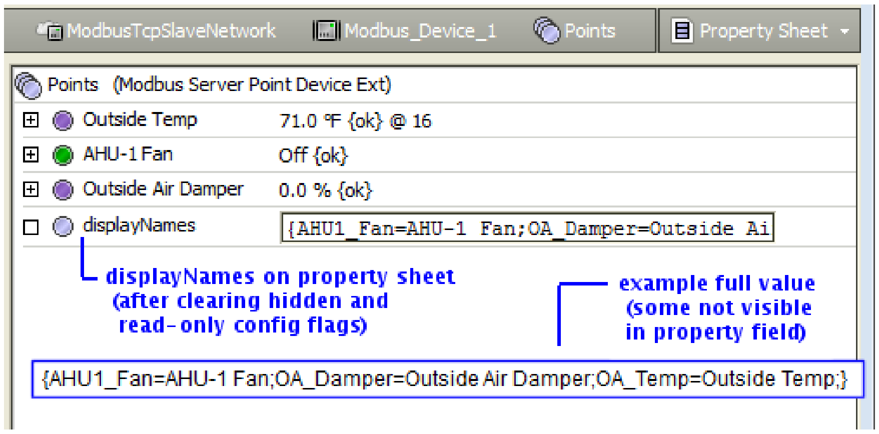
✓ Operator	No Audit
✓ Readonly	Composite
Confirm Required	Remove On Clone
Execute On Change	Metadata
Transient	Link Target
Summary	Non-Critical
No Run	User Defined 1
Fan In	User Defined 2
✓ Hidden	User Defined 3
Default On Clone	User Defined 4
Async	

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 7 *Example "Name Map" value of displayNames slot of parent with 3 entries*



ModbusTcpSlaveNetwork | Modbus_Device_1 | Points | Property Sheet

Points (Modbus Server Point Device Ext)

Outside Temp	71.0 °F {ok} @ 16
AHU-1 Fan	Off {ok}
Outside Air Damper	0.0 % {ok}
displayNames	{AHU1_Fan=AHU-1 Fan;OA_Damper=Outside Ai

displayNames on property sheet (after clearing hidden and read-only config flags)
example full value (some not visible in property field)

{AHU1_Fan=AHU-1 Fan;OA_Damper=Outside Air Damper;OA_Temp=Outside Temp;}

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.

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, page 14
- Set display names from slot sheets, page 14

Set a component display name

You can quickly set a display name for a component.

Prerequisites: Station opened in Workbench.

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

Step 1 In the Nav tree, expand the station's **Config** node to find the component, or locate the component in a property sheet.

Step 2 Right-click the component, and select **Set Display Name** from the popup menu.

Step 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.

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).

Prerequisites: Station opened in Workbench.

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

Step 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.

Step 2 Right-click the component, and select **Views→Slot Sheet** from the popup menu.

Step 3 In the slot sheet, double-click a slot to name.

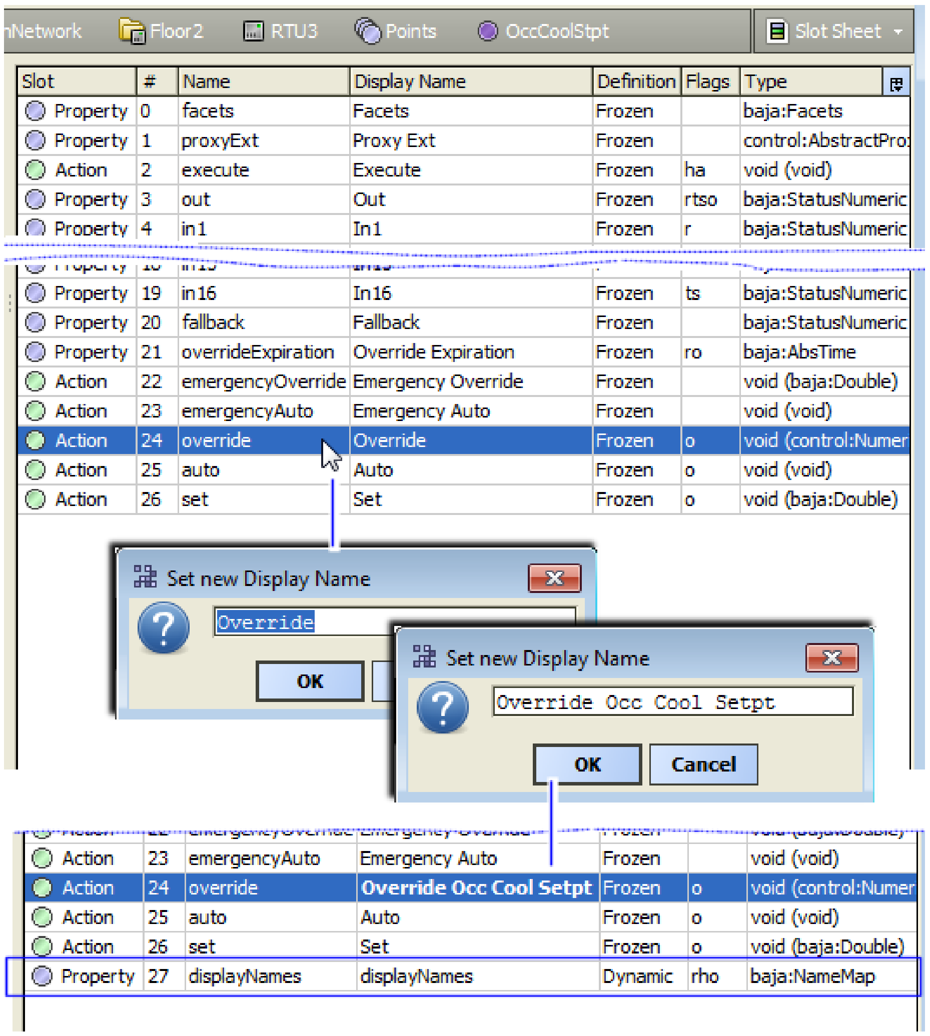
Step 4 In the **Set new Display Name** popup dialog, type the desired name to display for the child component (or slot, such as an action), and click **OK**.

Step 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 8 *Example first edit of slot display name for a control point slot (action)*



Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	facets	Facets	Frozen		baja:Facets
Property	1	proxyExt	Proxy Ext	Frozen		control:AbstractPro
Action	2	execute	Execute	Frozen	ha	void (void)
Property	3	out	Out	Frozen	rtso	baja:StatusNumeric
Property	4	in1	In1	Frozen	r	baja:StatusNumeric
Property	19	in16	In16	Frozen	ts	baja:StatusNumeric
Property	20	fallback	Fallback	Frozen		baja:StatusNumeric
Property	21	overrideExpiration	Override Expiration	Frozen	ro	baja:AbsTime
Action	22	emergencyOverride	Emergency Override	Frozen		void (baja:Double)
Action	23	emergencyAuto	Emergency Auto	Frozen		void (void)
Action	24	override	Override	Frozen	o	void (control:Numer
Action	25	auto	Auto	Frozen	o	void (void)
Action	26	set	Set	Frozen	o	void (baja:Double)

Set new Display Name

Override

OK

Set new Display Name

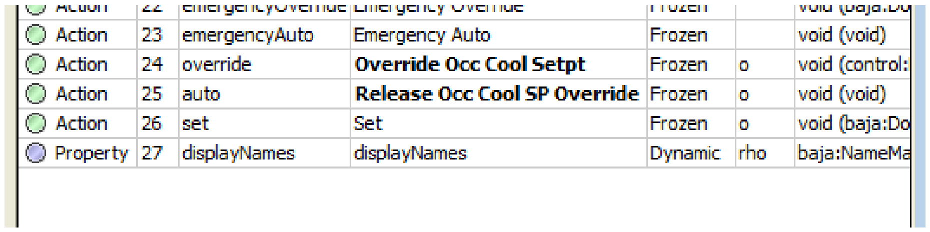
Override Occ Cool Setpt

OK Cancel

Action	23	emergencyAuto	Emergency Auto	Frozen		void (void)
Action	24	override	**Override Occ Cool Setpt**	Frozen	o	void (control:Numer
Action	25	auto	Auto	Frozen	o	void (void)
Action	26	set	Set	Frozen	o	void (baja:Double)
Property	27	displayNames	displayNames	Dynamic	rho	baja:NameMap

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 9   Second display name added, as shown in bolded Display Name text



Action	22	emergencyOverride	Emergency Override	Frozen		void (baja:Do
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.

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.

The following topics describe using the Batch Editor to batch edit or add display names:

- Using the Batch Editor to change existing display names, page 16
- Using the Batch Editor to add display names, page 19
- Using the Batch Editor to change slot flags, page 22
- Troubleshooting batch edited display names, page 24

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.

Prerequisites:

The station must be running and opened in Workbench, with the `program` module installed on the Niagar-aAX 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 us-ing 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. See ., page 22

Step 1 In Workbench with the station opened, access the **Batch Editor** (in the Nav tree, expand the Config, Services node and double-click **ProgramService**).

Step 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.

Step 3 Click **Edit Slot** to bring up the popup **Edit Slot** dialog.

Step 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.

Step 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.

Step 6 With the **New Value** field edited to the desired value(s), click **OK**.

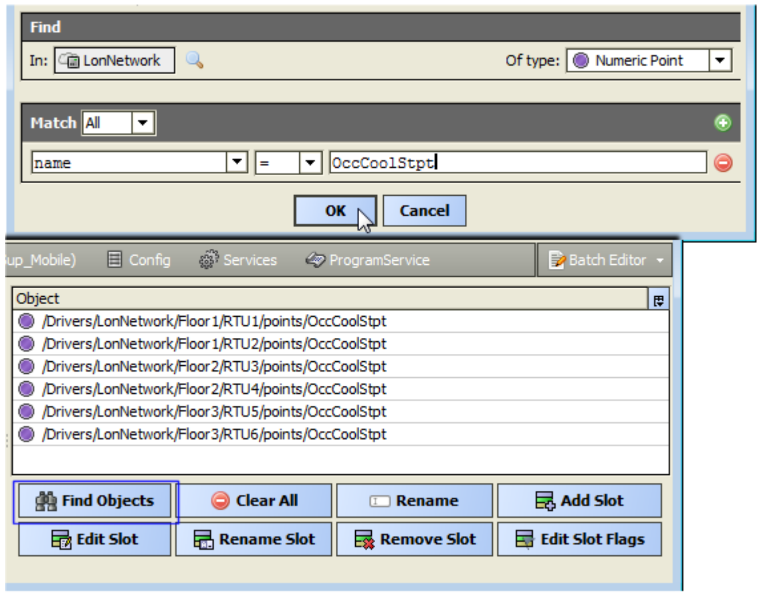
A **BatchEditor Results** popup replaces the **Edit Slot** dialog. It lists the edit slot results for each component listed in the Batch Editor.

Step 7 Click **OK** to close the **BatchEditor Results** dialog and return to the Batch Editor view.

Example batch edit of existing display names

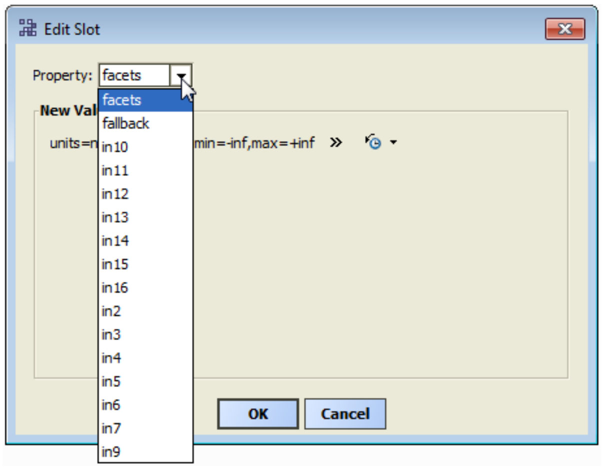
In this example, numeric points named "OccCoolStpt" under the station's LonNetwork are given a display name change.

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



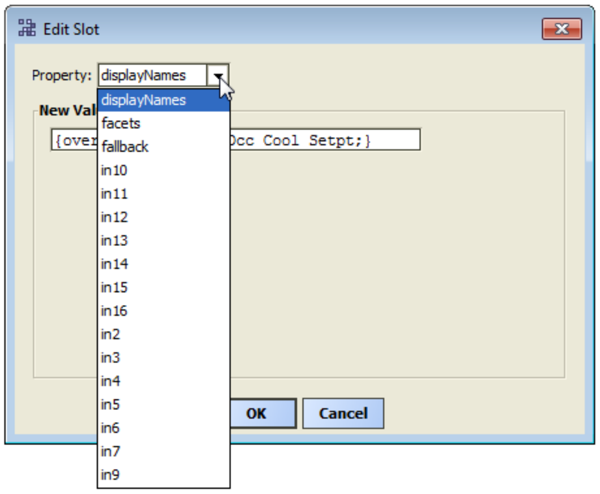
/Drivers/LonNetwork/Floor1/RTU1/points/OccCoolStpt
/Drivers/LonNetwork/Floor1/RTU2/points/OccCoolStpt
/Drivers/LonNetwork/Floor2/RTU3/points/OccCoolStpt
/Drivers/LonNetwork/Floor2/RTU4/points/OccCoolStpt
/Drivers/LonNetwork/Floor3/RTU5/points/OccCoolStpt
/Drivers/LonNetwork/Floor3/RTU6/points/OccCoolStpt

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

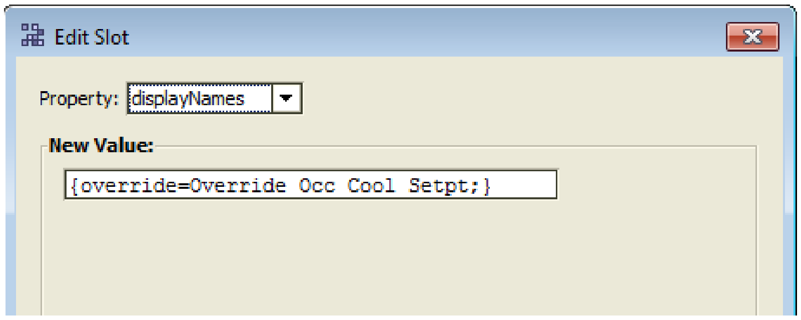


The **displayNames** slot is selected in the **Property** field. Note if all components listed in the **Batch Editor** have the "Readonly" flag set on their `displayNames` slot, the **Property** field is *not available* in the drop-down list, as shown in the preceding figure.

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.



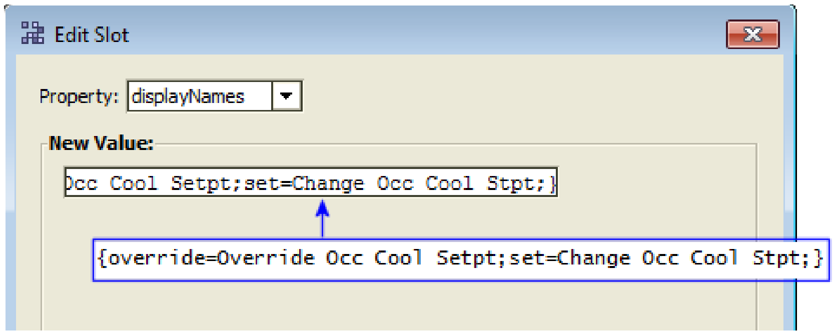
The **New Value** field in the **Edit Slot** dialog initially reflects the current `displayNames` slot value in the first component listed in the Batch Editor.



```
{override=Override Occ Cool Setpt;}
```

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

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

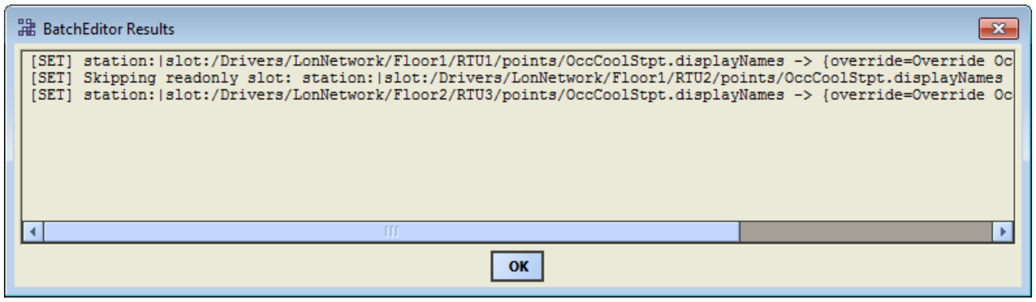


```
Occ Cool Setpt;set=Change Occ Cool Stpt;}
{override=Override Occ Cool Setpt;set=Change Occ Cool Stpt;}
```

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.



```
[SET] station:|slot:/Drivers/LonNetwork/Floor1/RTU1/points/OccCoolStpt.displayNames -> {override=Override Oc
[SET] Skipping readonly slot: station:|slot:/Drivers/LonNetwork/Floor1/RTU2/points/OccCoolStpt.displayNames
[SET] station:|slot:/Drivers/LonNetwork/Floor2/RTU3/points/OccCoolStpt.displayNames -> {override=Override Oc
```

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.

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.

Prerequisites:

The station must be running and opened in Workbench, with the `program` module installed on the NiagaraAX 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).

Step 1 In Workbench with the station opened, access the **Batch Editor** (in the Nav tree, expand the **Config**, **Services** node and double-click **ProgramService**).

Step 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.

Step 3 Click **Add Slot** to bring up the popup **Add Slot** dialog.

Step 4 In the **Add Slot** dialog, type `displayNames` in the **New Name** field.

Step 5 In the **New Type** field, ensure `baja` is selected on the left side, and click the right side drop-down control and select **NameMap**.

Step 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.

Step 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`.

Step 8 When finished in the **Add Slot** dialog, click **OK**.

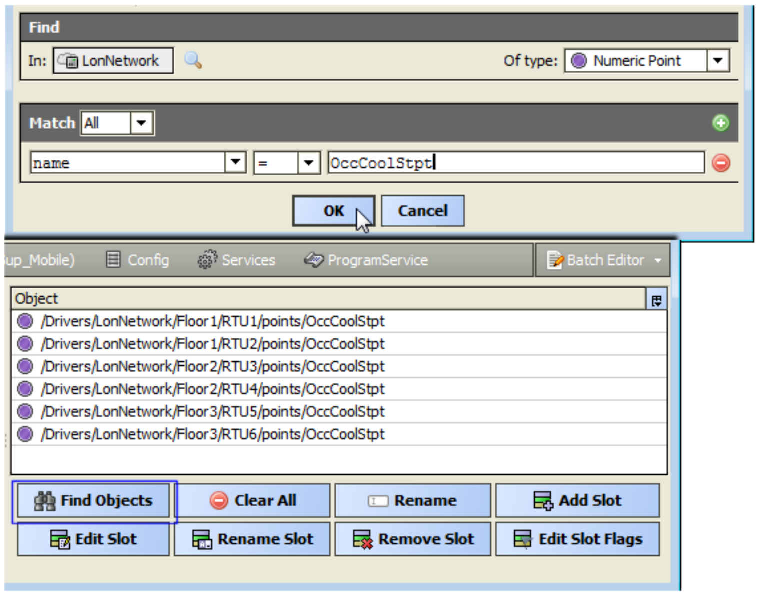
A **BatchEditor Results** popup dialog replaces the **Add Slot** dialog. It lists the add slot results for each component listed in the Batch Editor (if the **"Set if exists**" option was set); otherwise it lists the results only for those components listed that were previously without a `displayNames` slot.

Step 9 Click **OK** to close the **BatchEditor Results** dialog and return to the Batch Editor view.

Example batch edit to add display names

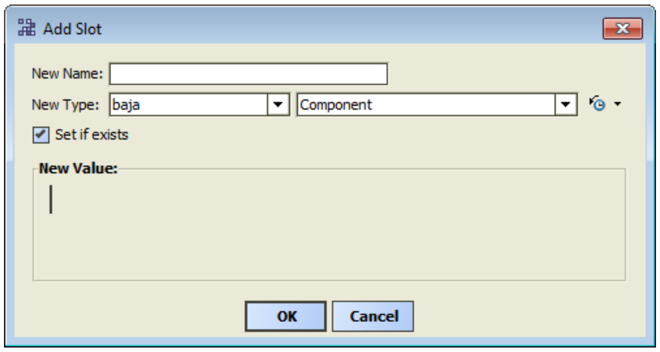
In this example, numeric points named "OccCoolStpt" under the station's **LonNetwork** are given a `displayNames` slot.

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



/Drivers/LonNetwork/Floor1/RTU1/points/OccCoolStpt
/Drivers/LonNetwork/Floor1/RTU2/points/OccCoolStpt
/Drivers/LonNetwork/Floor2/RTU3/points/OccCoolStpt
/Drivers/LonNetwork/Floor2/RTU4/points/OccCoolStpt
/Drivers/LonNetwork/Floor3/RTU5/points/OccCoolStpt
/Drivers/LonNetwork/Floor3/RTU6/points/OccCoolStpt

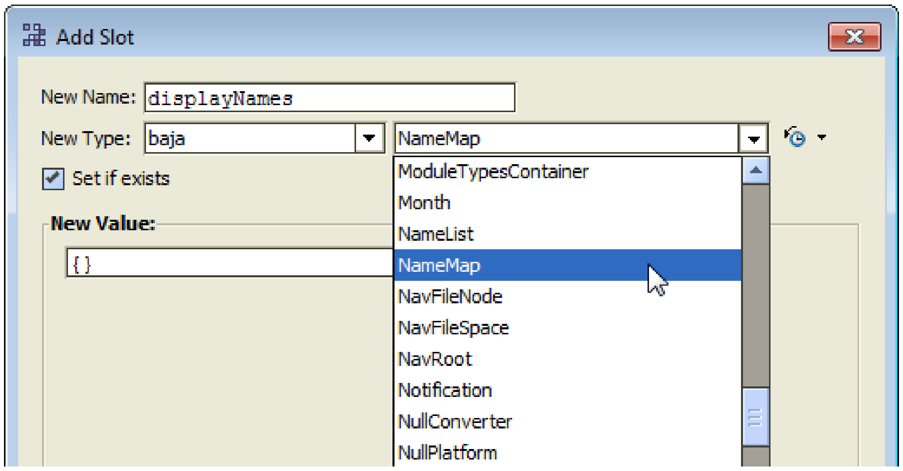
2. The **Add Slot** button is clicked, and the **Add Slot** popup dialog appears.



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

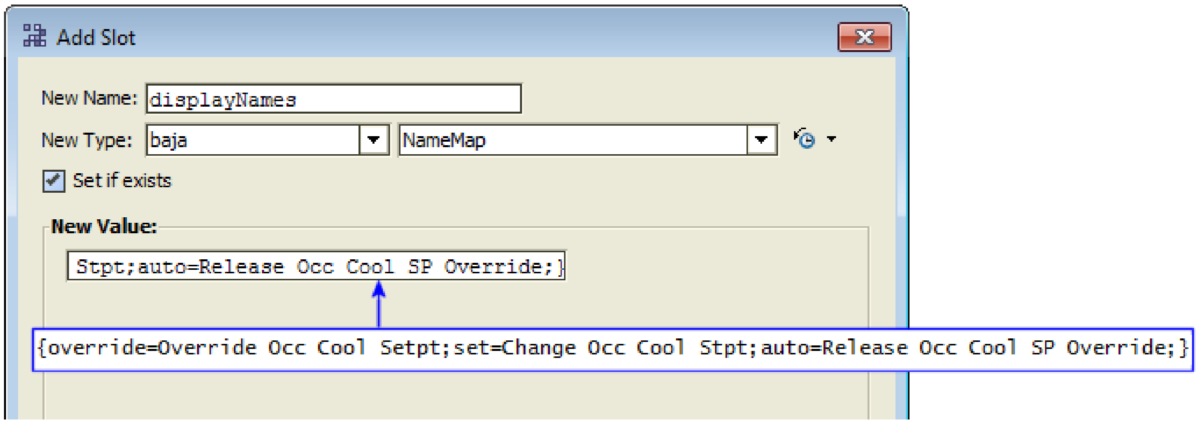
   NOTE: It must be entered exactly like this—plural.

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



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

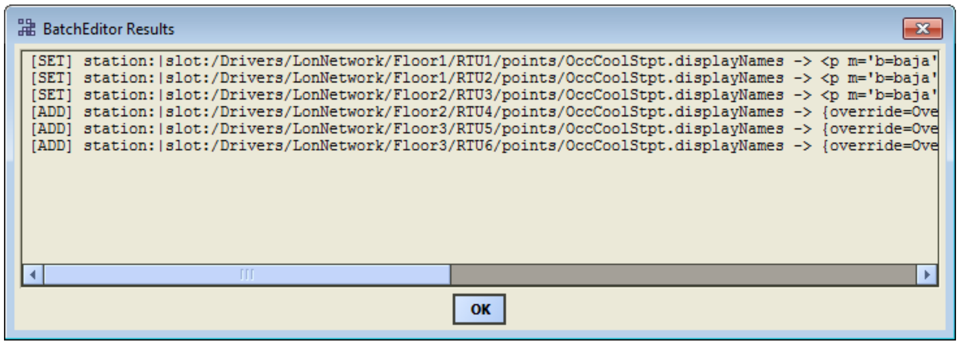
4. In the **New Value** field, the displayNames name map value is typed in.



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

   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.



```
[SET] station:|slot:/Drivers/LonNetwork/Floor1/RTU1/points/OccCoolStpt.displayNames -> <p m='b=baja'
[SET] station:|slot:/Drivers/LonNetwork/Floor1/RTU2/points/OccCoolStpt.displayNames -> <p m='b=baja'
[SET] station:|slot:/Drivers/LonNetwork/Floor2/RTU3/points/OccCoolStpt.displayNames -> <p m='b=baja'
[ADD] station:|slot:/Drivers/LonNetwork/Floor2/RTU4/points/OccCoolStpt.displayNames -> {override=Ove
[ADD] station:|slot:/Drivers/LonNetwork/Floor3/RTU5/points/OccCoolStpt.displayNames -> {override=Ove
[ADD] station:|slot:/Drivers/LonNetwork/Floor3/RTU6/points/OccCoolStpt.displayNames -> {override=Ove
```

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.

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.

Prerequisites:

The station must be running and opened in Workbench, with the `program` module installed on the NiagaraAX 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.

Step 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).

Step 2 Access the **Batch Editor** (in the Nav tree, expand the Config, Services node and double-click **ProgramService**).

Step 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.

Step 4 Click **Edit Slot Flags** to bring up the popup **Edit Slot Flags** dialog.

Step 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.

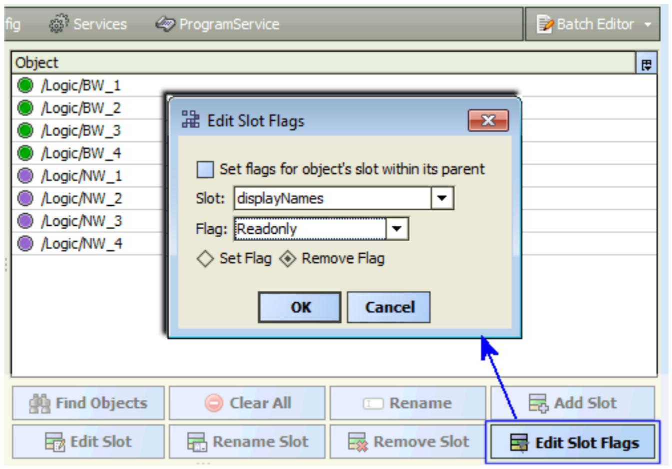
A **BatchEditor Results** popup replaces the **Edit Slot Flags** popup. It lists the results of the config flag batch edit.

Step 6 Click **OK** to close the results popup and return to the Batch Editor.

Example batch edit clearing of "Readonly" config flag on points

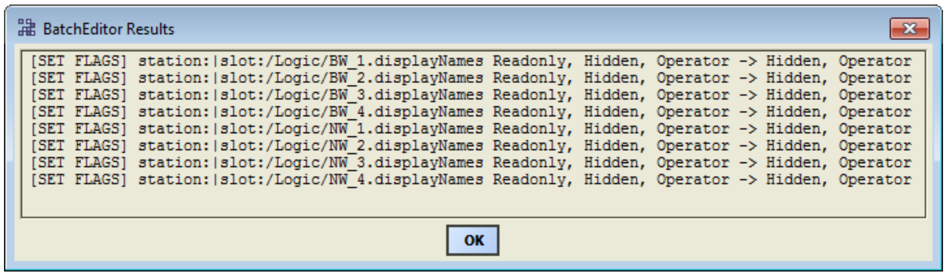
In this example, several points originally had action display names manually changed from their slot sheet. As a result, their `displayNames` slot has config flags set for "Readonly", "Hidden", and "Operator" (`rho`). In order to use the Batch Editor to further edit display names for the slots of these 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, `displayNames` is selected in the **Slot** field, and in **Flag** field the `Re-adonly` 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.



```
[SET FLAGS] station:|slot:/Logic/BW_1.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/BW_2.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/BW_3.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/BW_4.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/NW_1.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/NW_2.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/NW_3.displayNames Readonly, Hidden, Operator -> Hidden, Operator
[SET FLAGS] station:|slot:/Logic/NW_4.displayNames Readonly, Hidden, Operator -> Hidden, Operator
```

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.

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.

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 displayNames 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).

  For related details, see BNameMap syntax, page 13.
- 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`. Niagara looks specifically for a `displayNames` slot, and if misspelled, display names are not affected.

Chapter 2 Niagara Batch Editor

Topics covered in this chapter

- ♦ Batch Editor Interface
- ♦ Batch Editor quick reference
- ♦ Notes about using the Batch Editor
- ♦ Troubleshooting tips and examples

The Batch Editor performs specified operations on all selected items (objects). To select items, you drag and drop (or copy and paste) them into the Batch Editor's Object field. Or you can use the Bql Query Builder.

For example, if your installation has 150 points configured to go offnormal when a property exceeds a given limit, you could use the Batch Editor to change the limit on all objects at once. Otherwise, you would have to change the limit on each object's property sheet individually.

The Batch Editor requires the ProgramService in your Services container. If you don't have this service, copy it from the `program` palette.

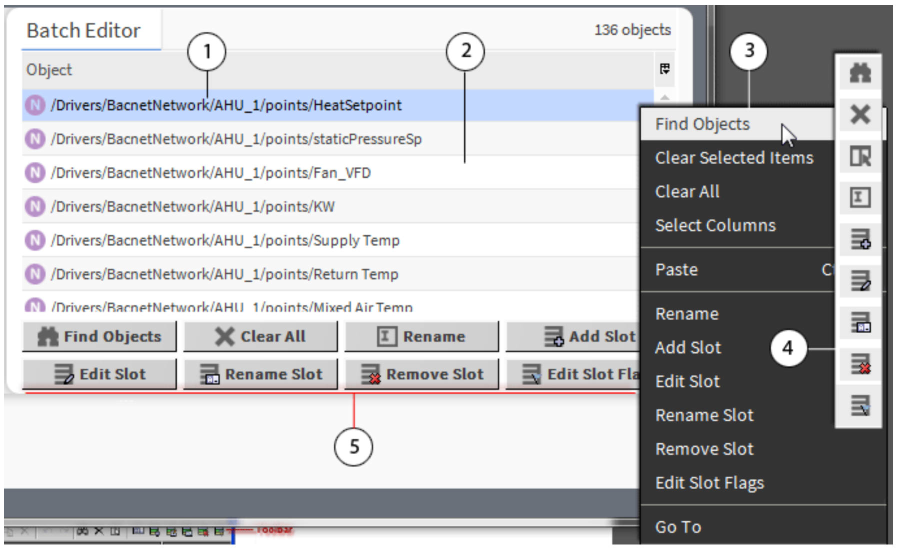
Batch Editor Interface

This topic describes the main areas and dialog boxes associated with the Batch Editor interface.

Main window

The main Batch Editor view is shown below and described in the following table.

Figure 10 Batch Editor tour


Batch Editor — 136 objects

Object
/Drivers/BacnetNetwork/AHU_1/points/HeatSetpoint
/Drivers/BacnetNetwork/AHU_1/points/staticPressureSp
/Drivers/BacnetNetwork/AHU_1/points/Fan_VFD
/Drivers/BacnetNetwork/AHU_1/points/KW
/Drivers/BacnetNetwork/AHU_1/points/Supply Temp
/Drivers/BacnetNetwork/AHU_1/points/Return Temp
/Drivers/BacnetNetwork/AHU_1/points/Mixed Air Temp

Name	Description
1. Found objects	The Batch Editor operates on all found objects. You can drag and drop (or copy and paste) single or selected objects onto the Object field, or use the Bql Query Builder to populate the field.
2. Object field	Displays each object including its path.

Name	Description
3. Popup menu	Right-clicking the Object field displays this menu, which provides the same functionality as the control buttons.
4. Toolbar	Provides the same functions as the control buttons and right-click menu.
5. Control buttons	Control buttons open dialog boxes or views that allow you to edit the selected items in the Found objects area.

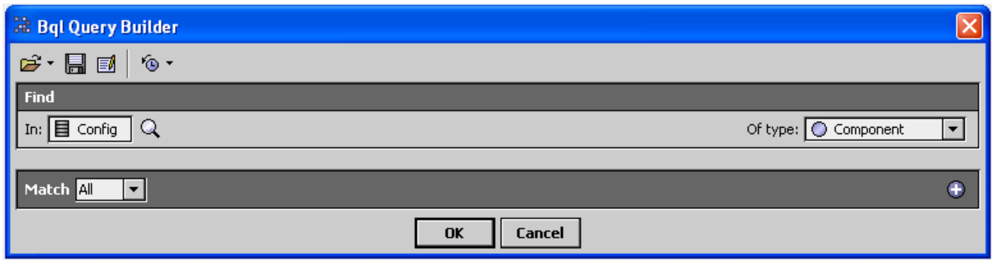
The following topics provide more details about the **Found Objects** area and the **Bql Query Builder**.

Found Objects area

There are three ways to populate the Object field in the Batch Editor main window.

- Drag and drop points (items) from the nav tree.
- Copy and paste points (items) from the nav tree.
- Click **Find Objects** and use the Bql Query Builder.

Figure 11 Bql Query Builder



– The 'In:' field allows you to define where to start searching in the nav tree.

– The 'Of Type' field lets you filter your search by type of component.

– The Match field works with the plus (⊕) to filter objects using search criteria.

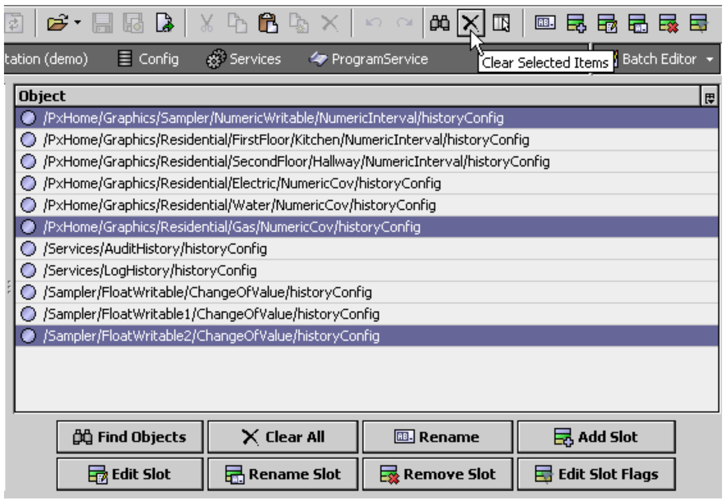
For more information about how to use the Bql Query Builder, see "About the Bql Query Builder" in the *Niagara Drivers Guide*.

Batch Editor Object field

There are two ways to remove items from the Batch Editor Object field.

- To clear selected items, hold down Ctrl and click to select items, then click the toolbar icon (✕) or use the right-click menu (**Clear Selected Items**).
- To remove all items, click the ✕ **Clear All** control button at the bottom of the view.

Figure 12 Clearing selected items from the Object field



Object
/PxHome/Graphics/Sampler/NumericWritable/NumericInterval/historyConfig
/PxHome/Graphics/Residential/FirstFloor/Kitchen/NumericInterval/historyConfig
/PxHome/Graphics/Residential/SecondFloor/Hallway/NumericInterval/historyConfig
/PxHome/Graphics/Residential/Electric/NumericCov/historyConfig
/PxHome/Graphics/Residential/Water/NumericCov/historyConfig
/PxHome/Graphics/Residential/Gas/NumericCov/historyConfig
/Services/AuditHistory/historyConfig
/Services/LogHistory/historyConfig
/Sampler/FloatWritable/ChangeOfValue/historyConfig
/Sampler/FloatWritable1/ChangeOfValue/historyConfig
/Sampler/FloatWritable2/ChangeOfValue/historyConfig

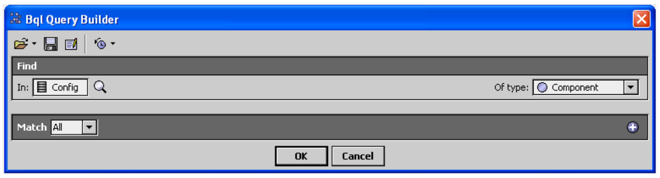
Batch Editor quick reference

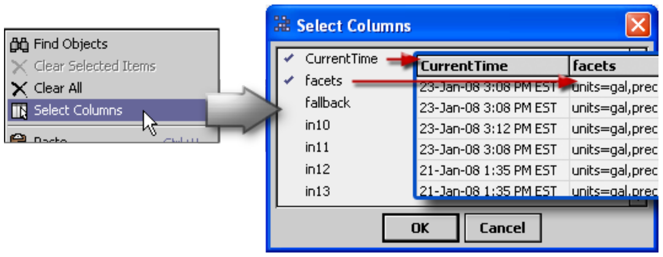
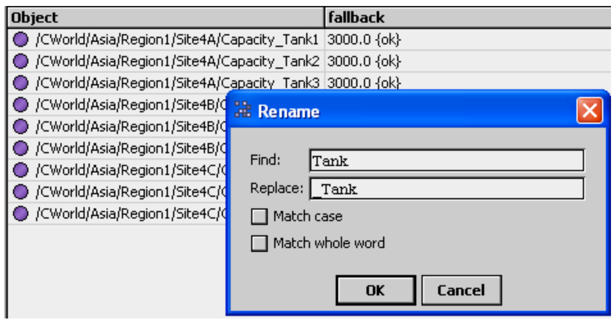
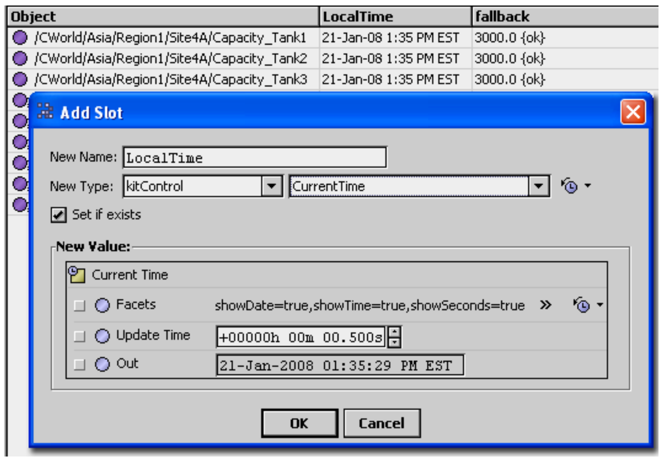
The Batch Editor performs the specified operation on all selected items (objects). To select items, you drag and drop (or copy and paste) them into the Batch Editor's Object field. Or you can use the Bql Query Builder.

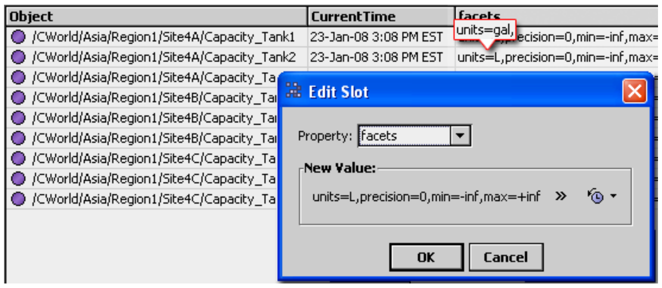
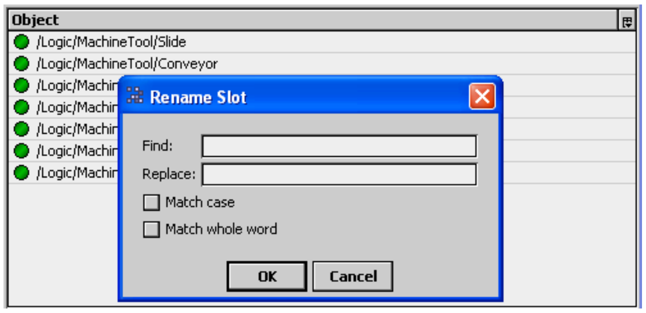
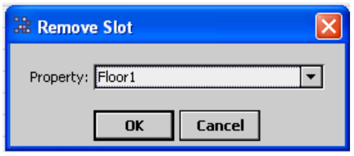
For example, if your installation has 150 points configured to go "offnormal" when a property exceeds a given limit, you could use the Batch Editor to change the limit on all objects at once. Otherwise, you would have to change the limit on each object's property sheet individually.

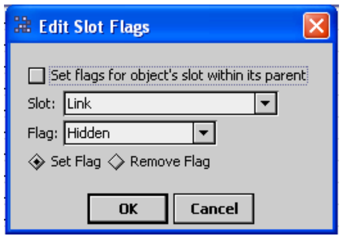
The Batch Editor requires the ProgramService in your Services container. If you don't have this service, copy it from the `program` palette.

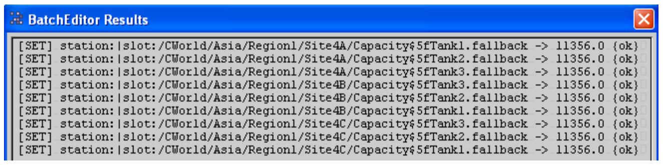
The following reference table describe ways to work with the Batch Editor.

Dialog Box or View	Description
Nav tree Add Objects directly to the **Bql Query Builder** view using the workbench nav tree.	• Drag and drop points (items) from the nav tree. • Click **Find Objects** and use the **Bql Query Builder.**
Bql Query Builder Add objects using the **Bql Query Builder.**  Bql Query Builder Find In: Config Of type: Component Match All OK Cancel	1. Click **Find Objects** and use the Bql Query Builder. 2. The 'In:' field allows you to define where to start searching in the nav tree. The 'Of Type' field lets you filter your search by type of component. The Match field works with the plus (⊕) to filter objects using search criteria. For more information about how to use the Bql Query Builder, see "About the Bql Query Builder" in the NiagaraAX Drivers Guide.
Add columns of information to the Object field using the popup menu and the **Select Columns** dialog box.	1. Populate the Object field with items to edit. You must have at least one item in the Object field.

Dialog Box or View	Description
 Find Objects Clear Selected Items Clear All Select Columns Select Columns CurrentTime facets fallback in10 in11 in12 in13 OK Cancel CurrentTime | facets 23-Jan-08 3:08 PM EST | units=gal,prec 23-Jan-08 3:08 PM EST | units=gal,prec 23-Jan-08 3:12 PM EST | units=gal,prec 23-Jan-08 3:08 PM EST | units=gal,prec 21-Jan-08 1:35 PM EST | units=gal,prec 21-Jan-08 1:35 PM EST | units=gal,prec	The Batch Editor lets you add different kinds of slots (objects) to the Object field. To avoid errors, make sure all objects are of the same type. 2. Click the Select Columns icon (◫) or right-click the field and click Select Columns. The Select Columns dialog box appears. 3. Click to mark the columns in the list, and click **OK** The image shows the right-click pop-up menu (on the left), the Select Columns dialog box overlaid by how the columns appear in the Object field.
Rename dialog box  Object | fallback /CWorld/Asia/Region1/Site4A/Capacity_Tank1 | 3000.0 {ok} /CWorld/Asia/Region1/Site4A/Capacity_Tank2 | 3000.0 {ok} /CWorld/Asia/Region1/Site4A/Capacity_Tank3 | 3000.0 {ok} /CWorld/Asia/Region1/Site4B/C /CWorld/Asia/Region1/Site4B/C /CWorld/Asia/Region1/Site4B/C /CWorld/Asia/Region1/Site4C/C /CWorld/Asia/Region1/Site4C/C /CWorld/Asia/Region1/Site4C/C Find: Tank Replace: _Tank	▤ **Rename** provides a find and replace feature. The Object field shows numeric points that have had their display names changed to add an underscore before the letters "Tank."
Add Slot dialog box  Object | LocalTime | fallback /CWorld/Asia/Region1/Site4A/Capacity_Tank1 | 21-Jan-08 1:35 PM EST | 3000.0 {ok} /CWorld/Asia/Region1/Site4A/Capacity_Tank2 | 21-Jan-08 1:35 PM EST | 3000.0 {ok} /CWorld/Asia/Region1/Site4A/Capacity_Tank3 | 21-Jan-08 1:35 PM EST | 3000.0 {ok} New Name: LocalTime New Type: kitControl | CurrentTime ☑ Set if exists Current Time Facets: showDate=true,showTime=true,showSeconds=true Update Time: +00000h 00m 00.500s Out: 21-Jan-2008 01:35:29 PM EST	The ▤ **Add Slot** feature lets you add the specified slot to all components in the Object field. • The values that appear in the New Value field depend on the selected slot type. • In the example above, a new slot named LocalTime is being added and configured. The new slot is defined as a CurrentTime component from the kitControl module. The Facets and Update Time properties in the New Value field box are configurable. • The 'Set if exists' check box allows you to change values for components that already have a LocalTime slot. If the check box is not selected, the CurrentTime properties are not changed for previously existing LocalTime slots.
Edit Slot dialog box	▤ **Edit Slot** can be used to edit any slot property. In the example shown here, the value of the facets property is being changed from units=gal (gallons) to units=L (Liters). The Object field shows the changed property in the facets column.

Dialog Box or View	Description
 Object / CurrentTime / facets: /CWorld/Asia/Region1/Site4A/Capacity_Tank1, 23-Jan-08 3:08 PM EST, units=gal,precision=0,min=-inf,max=; /CWorld/Asia/Region1/Site4A/Capacity_Tank2, 23-Jan-08 3:08 PM EST, units=L,precision=0,min=-inf,max= Edit Slot — Property: facets; New Value: units=L,precision=0,min=-inf,max=+inf	
Rename Slot dialog box  Rename Slot — Find: ; Replace: ; Match case; Match whole word	**Rename Slot** provides a find and replace feature.
Remove Slot dialog box  Remove Slot — Property: Floor1	**Remove Slot** makes it possible to remove all slots with the selected property. **NOTE:** There is no undo. Make sure you want to remove multiple slots before clicking OK.

Dialog Box or View	Description
Edit Slot Flags dialog box  Edit Slot Flags ☐ Set flags for object's slot within its parent Slot: Link Flag: Hidden ◈ Set Flag ◇ Remove Flag OK Cancel	You can batch edit slot flags. The following list describes the fields and properties of the **Edit Slot Flags** dialog box. • Set flags for object's slot within its parent This option sets flags for child objects. • Slot This drop-down list identifies the type of slot. • Flag This drop-down list identifies which flag to change. • Action This option identifies which action to take: – Set Flag or Remove Flag.

Batch Editor Results dialog box

BatchEditor Results
`[SET] station:|slot:/CWorld/Asia/Region1/Site4A/Capacity$5fTank1.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4A/Capacity$5fTank2.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4A/Capacity$5fTank3.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4B/Capacity$5fTank3.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4B/Capacity$5fTank2.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4B/Capacity$5fTank1.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4C/Capacity$5fTank3.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4C/Capacity$5fTank2.fallback -> 11356.0 {ok}`
`[SET] station:|slot:/CWorld/Asia/Region1/Site4C/Capacity$5fTank1.fallback -> 11356.0 {ok}`
	The **BatchEditor Results** dialog reports the action taken on each object.
	At the beginning of each row, the operation appears in square brackets, for example, `[SET]`. The object ORD comes next followed by an arrow (–>) and an indication of what changed or why no change occurred.

Open the Batch Editor

To open the Batch Editor view, open the Batch Editor view from the Program Service, as described in this topic.

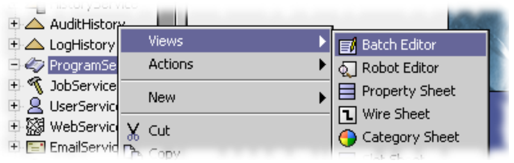
Step 1 Save and back up the station.

CAUTION: There is no undo. Should you make a mistake, it is always easier to reload a config.bog than to reconfigure the station.

Step 2 Do one of the following:

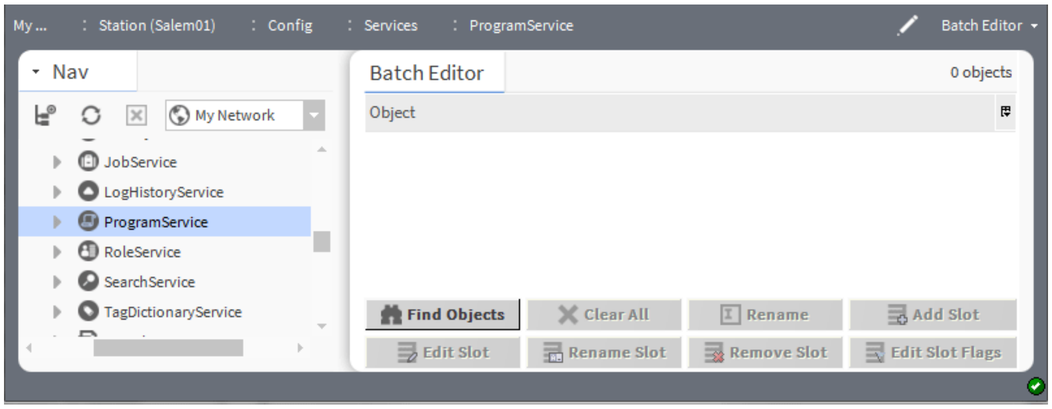
- Double-click the ProgramService container in the nav tree (Config\Services\ProgramService), or
- Right-click **ProgramService→→Views→Batch Editor**

Figure 13 Accessing the Batch Editor



An empty Object field appears.

Empty Batch Editor Objects field

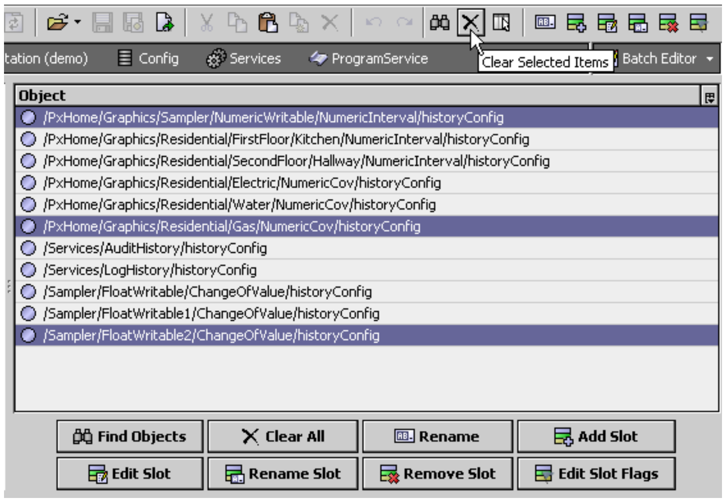


Clear the Batch Editor Object field

There are two ways to remove items from the Batch Editor Object field.

- To clear selected items, hold down Ctrl and click to select items, then click the toolbar icon (✕) or use the right-click menu (**Clear Selected Items**).
- To remove all items, click the ✕ **Clear All** control button at the bottom of the view.

Figure 14 *Clearing selected items from the Object field*



Object
/PxHome/Graphics/Sampler/NumericWritable/NumericInterval/historyConfig
/PxHome/Graphics/Residential/FirstFloor/Kitchen/NumericInterval/historyConfig
/PxHome/Graphics/Residential/SecondFloor/Hallway/NumericInterval/historyConfig
/PxHome/Graphics/Residential/Electric/NumericCov/historyConfig
/PxHome/Graphics/Residential/Water/NumericCov/historyConfig
/PxHome/Graphics/Residential/Gas/NumericCov/historyConfig
/Services/AuditHistory/historyConfig
/Services/LogHistory/historyConfig
/Sampler/FloatWritable/ChangeOfValue/historyConfig
/Sampler/FloatWritable1/ChangeOfValue/historyConfig
/Sampler/FloatWritable2/ChangeOfValue/historyConfig

Notes about using the Batch Editor

Following is a list of notes and recommended best practices related to using the Batch Editor.

- The operations run on the entire contents of the window—whether selected or not. You can highlight individual items for the purpose of selectively clearing them from the Object field, but you cannot highlight individual items for batch processing.
- To run a batch edit you may click the buttons below the Object field, use the toolbar at the top of the window, or right-click a blank area of the field and use the pop-up menu.
- You can populate the Object field by finding objects, dragging and dropping them from the nav tree, and copying and pasting them from the nav tree.
- Each find objects, drag and drop or copy and paste operation adds to the currently-selected objects. The Batch Editor does not automatically clear items from the Objects field.

- The Batch Editor runs on the items found in the current station with the ProgramService used to launch the editor. Attempting to operate on objects left over from a previous station typically causes a "not found" error.
- You will find that some slots cannot be renamed or removed, unless the objects also exist on the current station. These slots appear as "frozen" on the property sheet.
- Always back up the station before you start batch editing slots.

TIP:

- Edit a single object first. You can easily drag and drop it onto the Object field and test the change. Make sure you are happy with the results before you find the other objects and change them.
- Resources in workbench:
  - Use the slot sheet to view object names and flags.
  - Use the Bajadoc to locate the name of the precise component to change.

Troubleshooting tips and examples

Following, are some solutions for a few common situations and several examples of using the Batch Editor.

Troubleshooting tips

- If some items update while others generate errors in the BatchEditor Results box, make sure you are editing like items.
- The Batch Editor will let you populate the Object field with all types of slots, but you may not want to run a batch edit on all of them.
- A good practice is to click Clear All before starting a batch edit session.

NOTE: You cannot batch edit frozen slots

Examples

The following tasks topics are examples of how you can use the Batch Editor.

Hide action slot on BooleanWritables

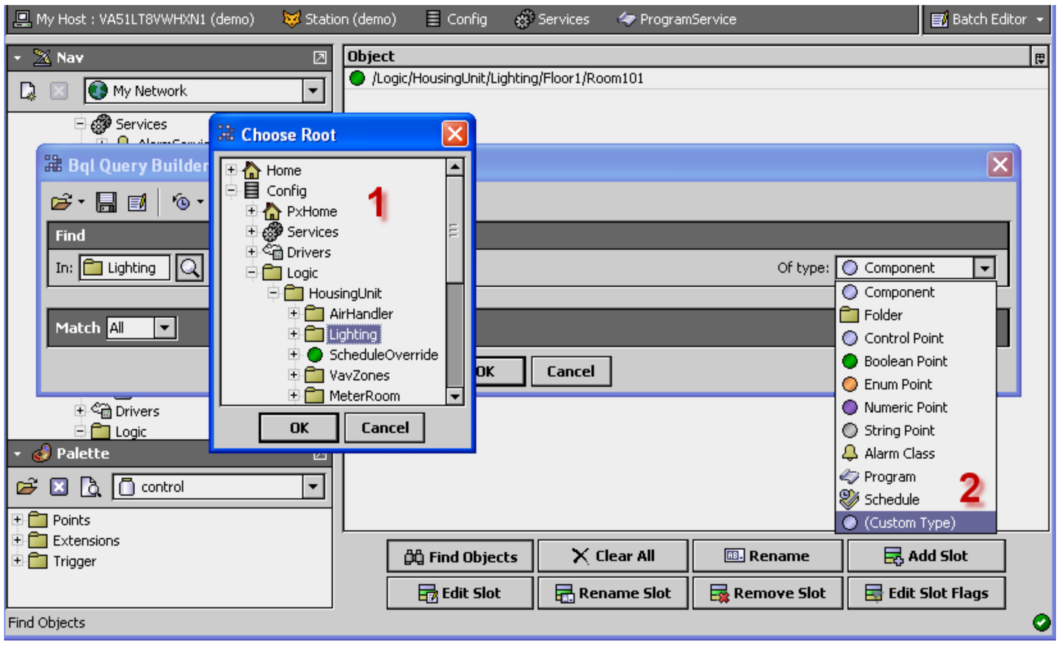
The ability to batch edit slots is the primary use of the Batch Editor. For NiagaraAX-3.6 and later systems (Workbench and JACE controllers), you can also edit slot flags (Config Flags). For example, you can set or clear the "Operator" flag on slots, or set or clear the "Hidden" flag on slots.

Prerequisites:

In this example, we will hide the emergencyActive slot on a number of BooleanWritable components. Step numbers correspond to the numbers in the graphics.

Step 1 To locate the components to edit, we'll use the Bql Query Builder (Click **Find Objects** to display the Bql Query Builder) and narrow the search by drilling down in the nav tree.

Figure 15 Narrowing the search

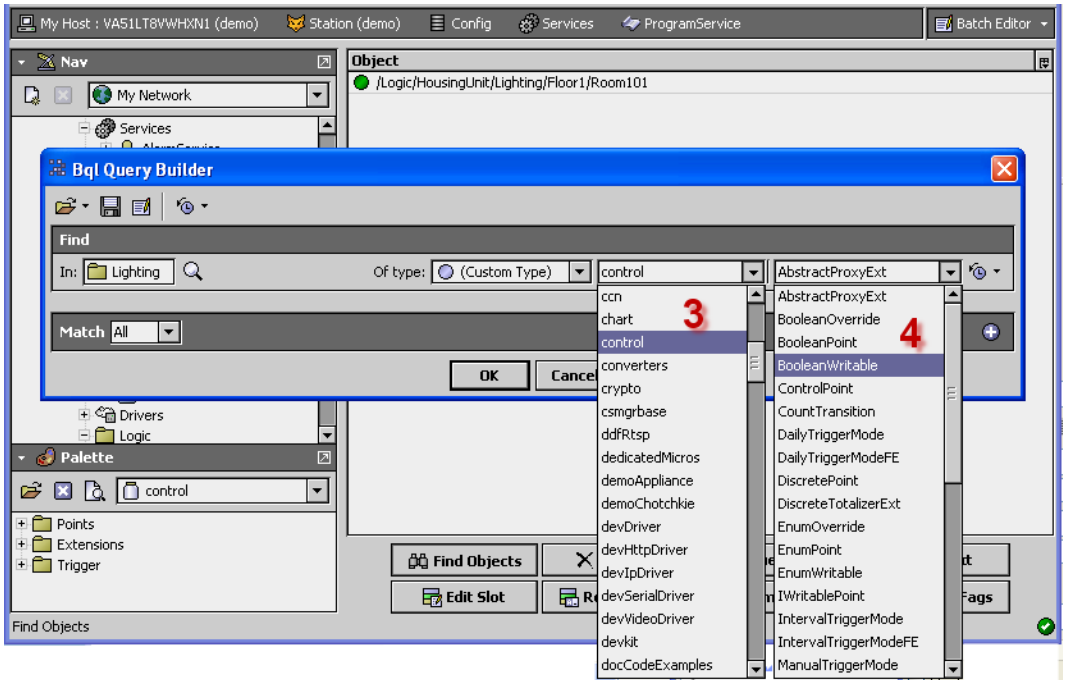


Step 2 To further narrow the search we'll select the module and component by choosing (Custom Type) from the 'Of Type:' drop-down list.

This opens two additional lists: one for the module and the other for the component.

Step 3 Knowing that a BooleanWriteable is in the control module, we choose control from the drop-down list.

Figure 16 Choosing component type

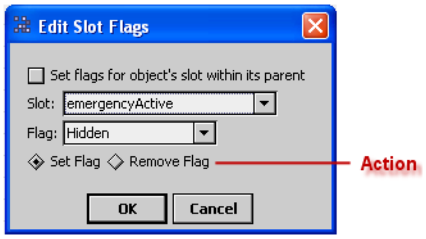


Step 4 Next we choose the component type from the component drop-down list and click **OK**.

NOTE: To find the module and component type, refer to the 'Type' column on the slot sheet. The module and component type are displayed in the format: module:component.

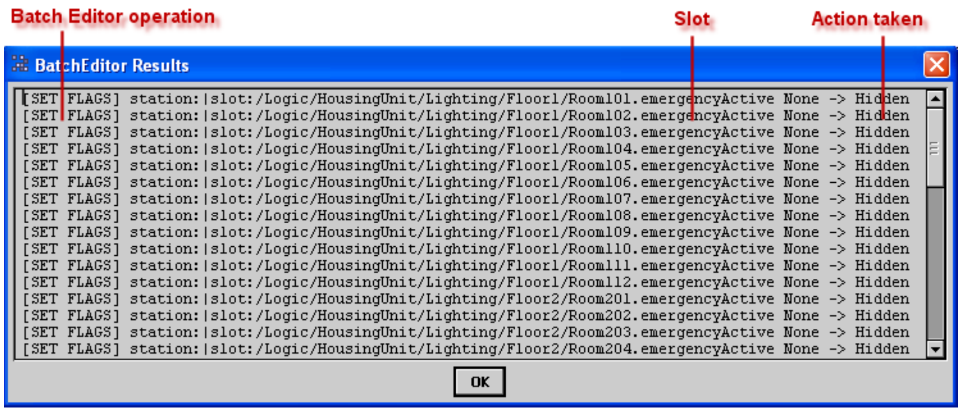
Step 5 Finally, we click **Edit Slot Flags**, set Slot to emergencyActive, set Flag to Hidden, set action to Set Flag and click **OK**.

Figure 17 *Edit Slot Flags dialog box*



All the slot flags change and the Batch Editor displays the results.

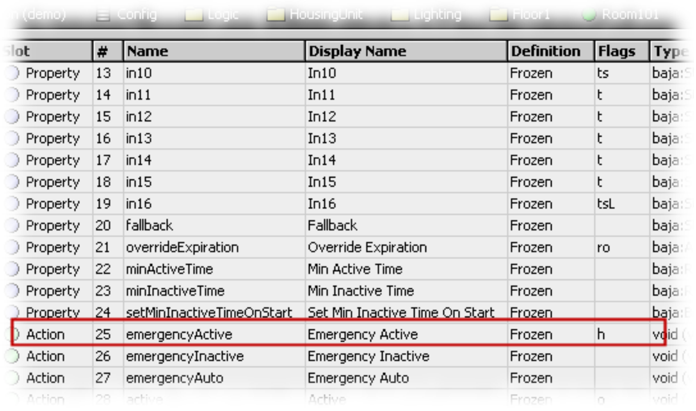
Figure 18 *BatchEditor Results*



```
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room101.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room102.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room103.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room104.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room105.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room106.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room107.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room108.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room109.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room110.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room111.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor1/Room112.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor2/Room201.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor2/Room202.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor2/Room203.emergencyActive None -> Hidden
[SET FLAGS] station:|slot:/Logic/HousingUnit/Lighting/Floor2/Room204.emergencyActive None -> Hidden
```

Step 6 To confirm the change, check the slot sheet for one of a changed component.

Figure 19 *Fragment of a slot sheet showing the change*



lot	#	Name	Display Name	Definition	Flags	Type
Property	13	in10	In10	Frozen	ts	baja:
Property	14	in11	In11	Frozen	t	baja:
Property	15	in12	In12	Frozen	t	baja:
Property	16	in13	In13	Frozen	t	baja:
Property	17	in14	In14	Frozen	t	baja:
Property	18	in15	In15	Frozen	t	baja:
Property	19	in16	In16	Frozen	tsL	baja:
Property	20	fallback	Fallback	Frozen		baja:
Property	21	overrideExpiration	Override Expiration	Frozen	ro	baja:
Property	22	minActiveTime	Min Active Time	Frozen		baja:
Property	23	minInactiveTime	Min Inactive Time	Frozen		baja:
Property	24	setMinInactiveTimeOnStart	Set Min Inactive Time On Start	Frozen		baja:
Action	25	emergencyActive	Emergency Active	Frozen	h	void (
Action	26	emergencyInactive	Emergency Inactive	Frozen		void (
Action	27	emergencyAuto	Emergency Auto	Frozen		void (
Action	28	active	Active	Frozen	o	void (

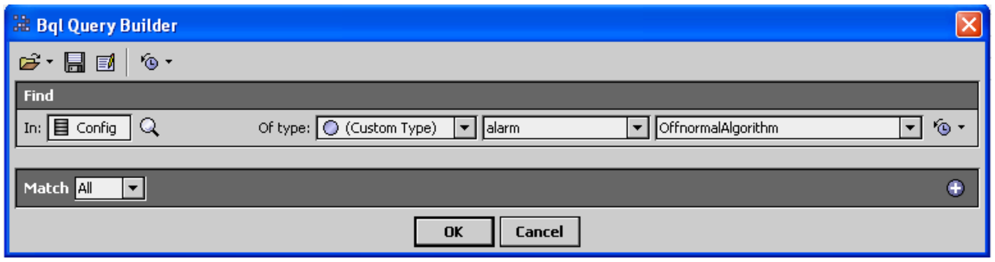
In the image above, "h" in the Flags column indicates that the hidden flag for slot emergencyActive has been set.

Set the offnormal high limit on alarm extensions

In this example, we will change the temperature high limit from 100° to 95°. We'll also demonstrate how to use the right-click pop-up menu to selectively clear items that you don't want to include in the change.

Step 1 In the Batch Editor Bql Query Builder (double-lick ProgramService and click **Find Objects**) configure 'Of Type', (Custom Type) as follows.

Figure 20 Bql Query Builder configured to locate OffnormalAlgorithm

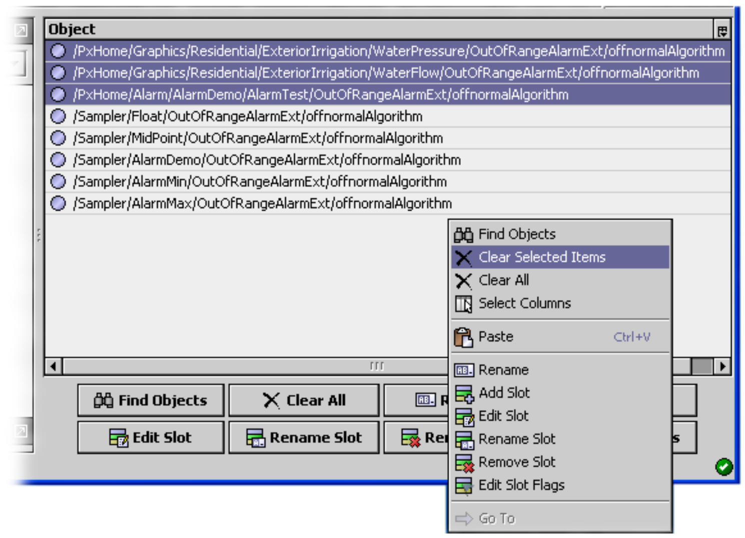


In: Config
Of type: (Custom Type) alarm OffnormalAlgorithm
Match All

The OffnormalAlgorithm is the component used to define the conditions that trigger an alarm.

Step 2 In this step we select and clear from the Object field the objects we do not want to change.

Figure 21 Selected objects to be removed

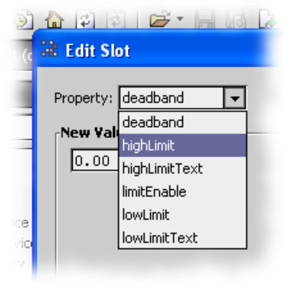


Object
/PxHome/Graphics/Residential/ExteriorIrrigation/WaterPressure/OutOfRangeAlarmExt/offnormalAlgorithm
/PxHome/Graphics/Residential/ExteriorIrrigation/WaterFlow/OutOfRangeAlarmExt/offnormalAlgorithm
/PxHome/Alarm/AlarmDemo/AlarmTest/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/Float/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/MidPoint/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmDemo/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmMin/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmMax/OutOfRangeAlarmExt/offnormalAlgorithm

The quickest way to clear items from the Object field is to click the Clear Selected Items icon in the toolbar. If you're using the pop-up menu, right-click in the blank area of the Object field otherwise you may inadvertently deselect an object.

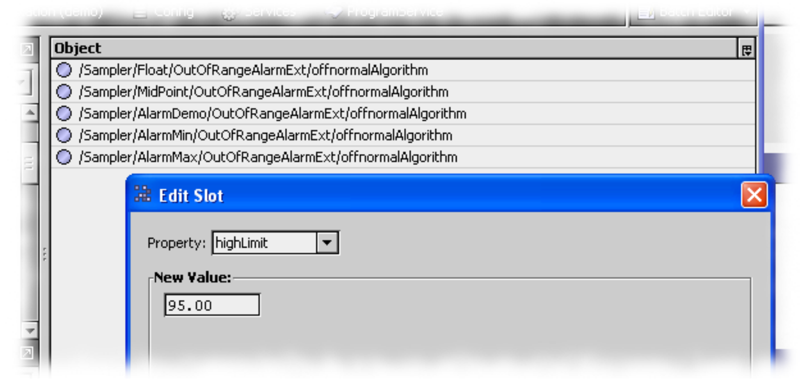
Step 3 Click **Edit Slot** and choose the highLimit property from the Property drop-down menu.

Figure 22 *Choosing the property to edit*



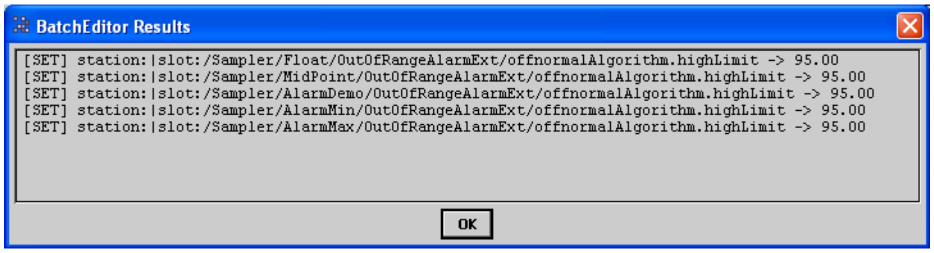
Step 4 Set the New Value field to 95 and click **OK**.

Figure 23 *Setting the property*


Object
/Sampler/Float/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/MidPoint/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmDemo/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmMin/OutOfRangeAlarmExt/offnormalAlgorithm
/Sampler/AlarmMax/OutOfRangeAlarmExt/offnormalAlgorithm
Edit Slot
Property: highLimit
New Value:
95.00

The Batch Editor changes the objects. This may take a few seconds. Then it displays the results.

Figure 24 *Results*

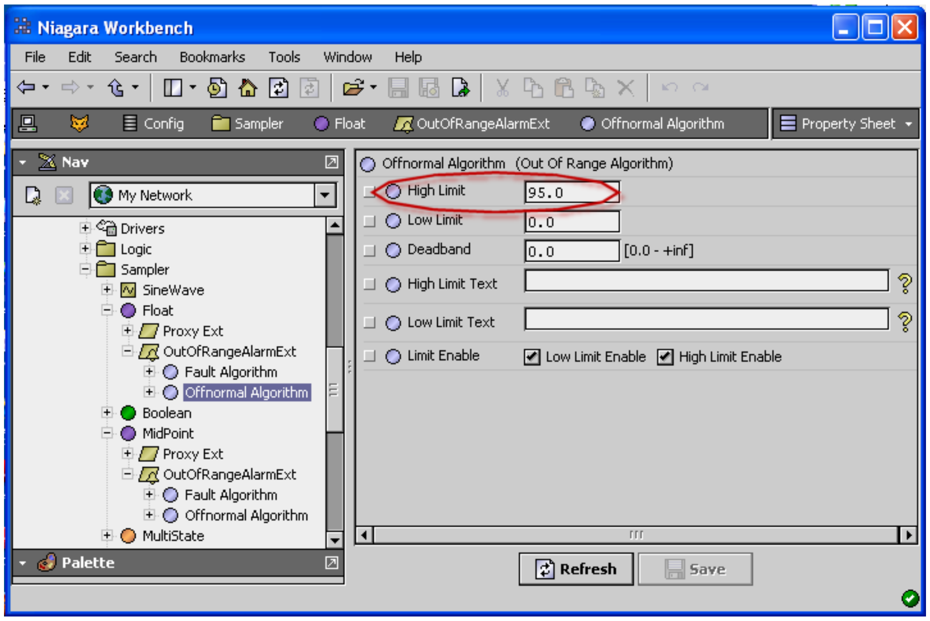


```
[SET] station:|slot:/Sampler/Float/OutOfRangeAlarmExt/offnormalAlgorithm.highLimit -> 95.00
[SET] station:|slot:/Sampler/MidPoint/OutOfRangeAlarmExt/offnormalAlgorithm.highLimit -> 95.00
[SET] station:|slot:/Sampler/AlarmDemo/OutOfRangeAlarmExt/offnormalAlgorithm.highLimit -> 95.00
[SET] station:|slot:/Sampler/AlarmMin/OutOfRangeAlarmExt/offnormalAlgorithm.highLimit -> 95.00
[SET] station:|slot:/Sampler/AlarmMax/OutOfRangeAlarmExt/offnormalAlgorithm.highLimit -> 95.00
```

Step 5 To close the BatchEditor Results, click **OK**.

Step 6 Check the property sheet for one of the points to ensure that the change was made.

Figure 25 Property sheet



Offnormal Algorithm (Out Of Range Algorithm)

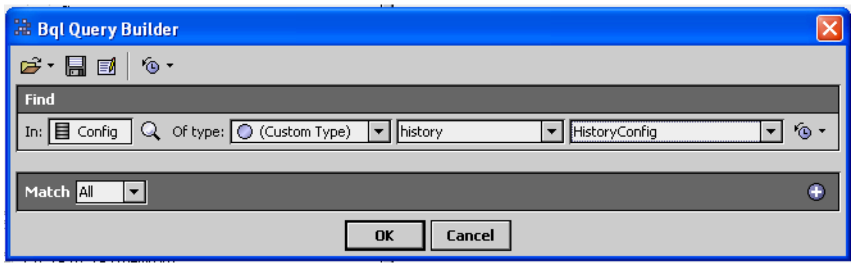
High Limit	95.0
Low Limit	0.0
Deadband	0.0 [0.0 - +inf]
High Limit Text	
Low Limit Text	
Limit Enable	✓ Low Limit Enable ✓ High Limit Enable

Increase history record count

In this example we will expand the capacity of the history databases that currently hold 500 records to allow them to hold 600 records. First we will do a wide search for all historyConfig containers, starting from the Config folder in the nav tree. Then, we will clear the containers configured to hold less than 500 records. And finally, we will change the capacity property on the remaining historyConfig containers.

Step 1 Using the Bql Query Builder, search for all historyConfig container slots.

Figure 26 Bql Query Builder



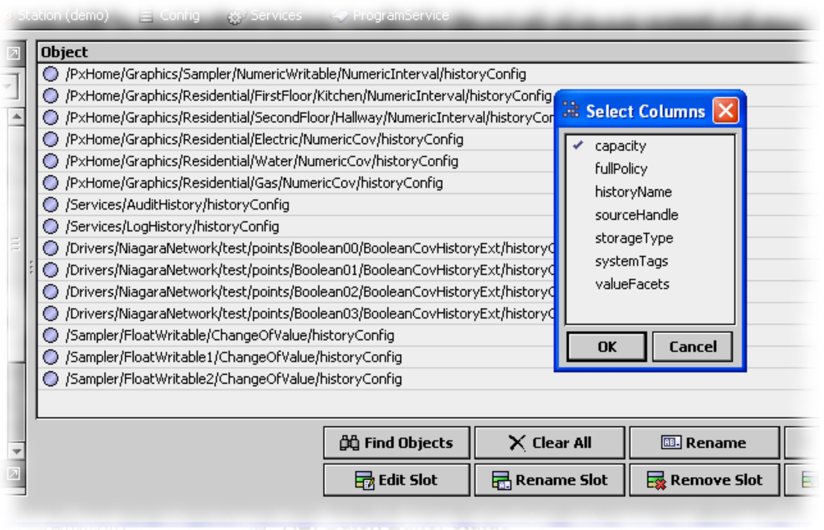
In: Config Of type: (Custom Type) history HistoryConfig

Match All

Step 2 Expand the window slightly so you can see an additional column.

Step 3 To add a capacity column to the Object field, click the Select Columns icon (▥) in the toolbar.

Figure 27 Adding the capacity column

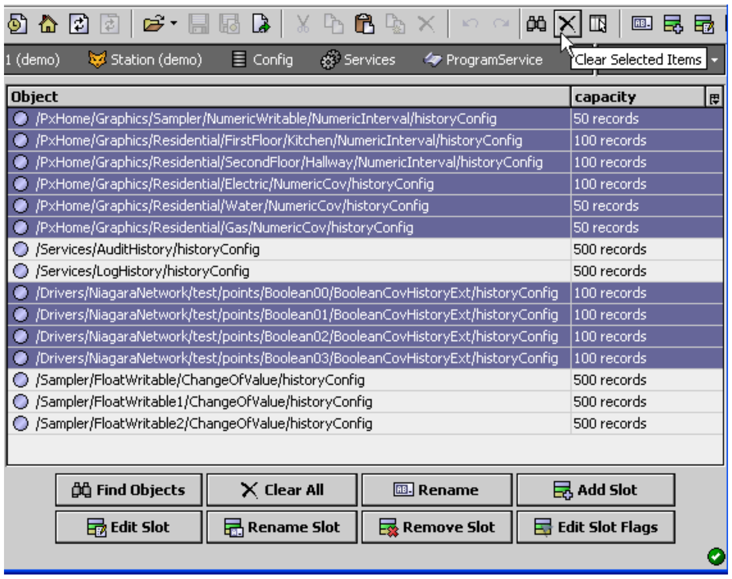


Object
/PxHome/Graphics/Sampler/NumericWritable/NumericInterval/historyConfig
/PxHome/Graphics/Residential/FirstFloor/Kitchen/NumericInterval/historyConfig
/PxHome/Graphics/Residential/SecondFloor/Hallway/NumericInterval/historyCon
/PxHome/Graphics/Residential/Electric/NumericCov/historyConfig
/PxHome/Graphics/Residential/Water/NumericCov/historyConfig
/PxHome/Graphics/Residential/Gas/NumericCov/historyConfig
/Services/AuditHistory/historyConfig
/Services/LogHistory/historyConfig
/Drivers/NiagaraNetwork/test/points/Boolean00/BooleanCovHistoryExt/history
/Drivers/NiagaraNetwork/test/points/Boolean01/BooleanCovHistoryExt/history
/Drivers/NiagaraNetwork/test/points/Boolean02/BooleanCovHistoryExt/history
/Drivers/NiagaraNetwork/test/points/Boolean03/BooleanCovHistoryExt/history
/Sampler/FloatWritable/ChangeOfValue/historyConfig
/Sampler/FloatWritable1/ChangeOfValue/historyConfig
/Sampler/FloatWritable2/ChangeOfValue/historyConfig

- ✓ capacity
- fullPolicy
- historyName
- sourceHandle
- storageType
- systemTags
- valueFacets

Step 4 To clear the historyConfig slots configured to hold less than 500 records, click the Clear Selected Items icon in the toolbar.

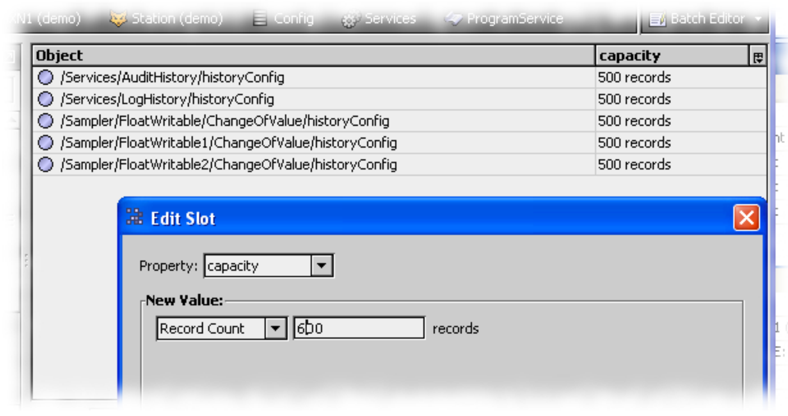
Figure 28 Highlighted historyConfig container slots to be cleared



Object	capacity
/PxHome/Graphics/Sampler/NumericWritable/NumericInterval/historyConfig	50 records
/PxHome/Graphics/Residential/FirstFloor/Kitchen/NumericInterval/historyConfig	100 records
/PxHome/Graphics/Residential/SecondFloor/Hallway/NumericInterval/historyConfig	100 records
/PxHome/Graphics/Residential/Electric/NumericCov/historyConfig	100 records
/PxHome/Graphics/Residential/Water/NumericCov/historyConfig	50 records
/PxHome/Graphics/Residential/Gas/NumericCov/historyConfig	50 records
/Services/AuditHistory/historyConfig	500 records
/Services/LogHistory/historyConfig	500 records
/Drivers/NiagaraNetwork/test/points/Boolean00/BooleanCovHistoryExt/historyConfig	100 records
/Drivers/NiagaraNetwork/test/points/Boolean01/BooleanCovHistoryExt/historyConfig	100 records
/Drivers/NiagaraNetwork/test/points/Boolean02/BooleanCovHistoryExt/historyConfig	100 records
/Drivers/NiagaraNetwork/test/points/Boolean03/BooleanCovHistoryExt/historyConfig	100 records
/Sampler/FloatWritable/ChangeOfValue/historyConfig	500 records
/Sampler/FloatWritable1/ChangeOfValue/historyConfig	500 records
/Sampler/FloatWritable2/ChangeOfValue/historyConfig	500 records

Step 5 Click **Edit Slot**, change the capacity property to 600 records and click **OK**.

Figure 29   Capacity changed to 600 records



Object	capacity
/Services/AuditHistory/historyConfig	500 records
/Services/LogHistory/historyConfig	500 records
/Sampler/FloatWritable/ChangeOfValue/historyConfig	500 records
/Sampler/FloatWritable1/ChangeOfValue/historyConfig	500 records
/Sampler/FloatWritable2/ChangeOfValue/historyConfig	500 records

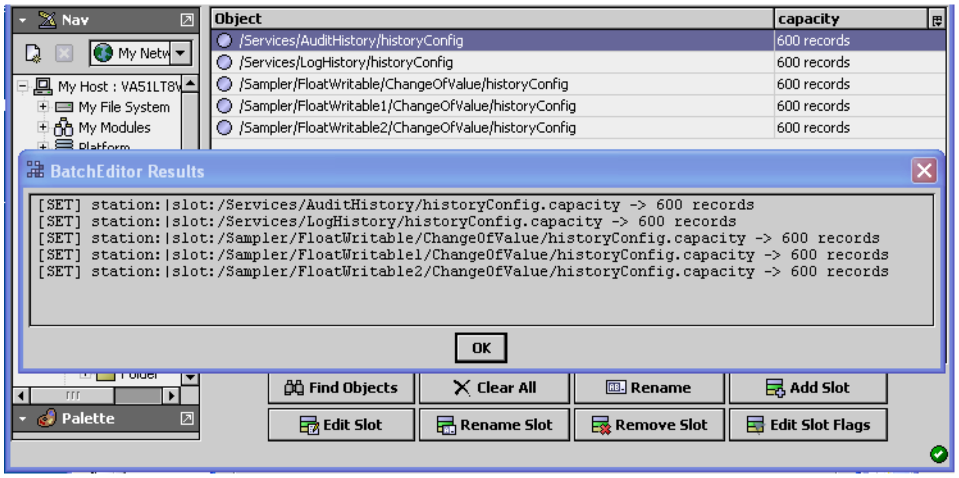
Edit Slot

Property: capacity

New Value: Record Count 600 records

The results indicate the change was made and the capacity column changes to reflect the change.

Figure 30   Capacities changed



Object	capacity
/Services/AuditHistory/historyConfig	600 records
/Services/LogHistory/historyConfig	600 records
/Sampler/FloatWritable/ChangeOfValue/historyConfig	600 records
/Sampler/FloatWritable1/ChangeOfValue/historyConfig	600 records
/Sampler/FloatWritable2/ChangeOfValue/historyConfig	600 records

BatchEditor Results

```
[SET] station:|slot:/Services/AuditHistory/historyConfig.capacity -> 600 records
[SET] station:|slot:/Services/LogHistory/historyConfig.capacity -> 600 records
[SET] station:|slot:/Sampler/FloatWritable/ChangeOfValue/historyConfig.capacity -> 600 records
[SET] station:|slot:/Sampler/FloatWritable1/ChangeOfValue/historyConfig.capacity -> 600 records
[SET] station:|slot:/Sampler/FloatWritable2/ChangeOfValue/historyConfig.capacity -> 600 records
```

Chapter 3 Niagara R2 to NiagaraAX via oBIX

Topics covered in this chapter

♦ R2 to AX oBIX Overview
♦ R2 station engineering
♦ AxSupervisor engineering

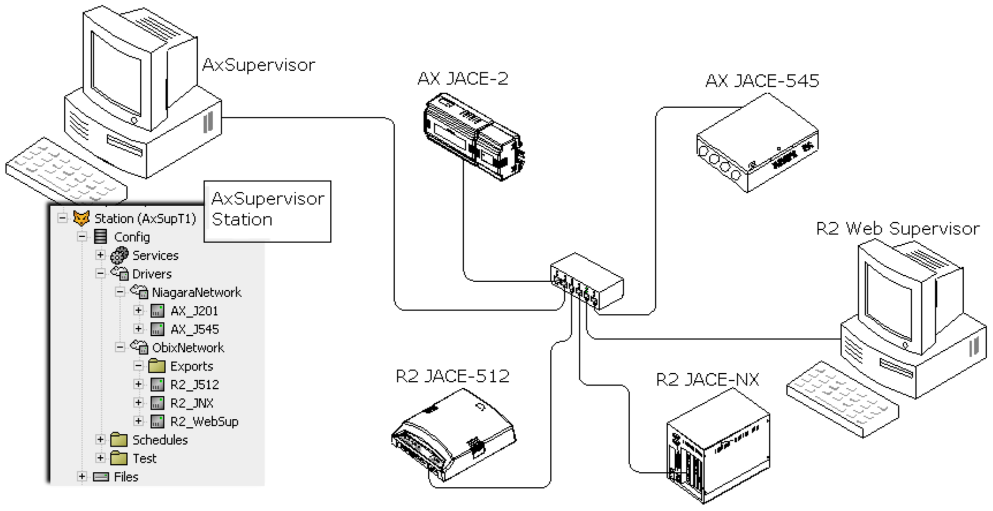
This document provides information on integrating Niagara R2 stations into a NiagaraAX system using oBIX technology, and assumes that you are knowledgeable about the Niagara engineering used in both system types (often abbreviated simply "R2" and "AX"). Other reference details about Niagara oBIX implementations can be found in the *NiagaraAX oBIX Guide* 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 / NiagaraAX 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 NiagaraAX-3.1 and later, an AX station 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, an AxSupervisor will be added to a job that already has R2 JACEs and an R2 Web Supervisor, along with additional AX JACEs. A possible eventual goal of the AxSupervisor is to replace the R2 Web Supervisor. This document focuses on the AxSupervisor, and notes areas where R2 to AX "switchover" has limitations, or may require additional engineering.

Figure 31 *AxSupervisor access to R2 stations and AX JACEs is using different network types*



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 AX client side, in coordination with an AX feature to hide rarely accessed (internal) properties. See "About Discover "Include" options" on page 2-10 for related details.

- The AxSupervisor host must be running NiagaraAX 3.1.31 or later, have the installed modules:
  - `obix` 3.3.22 or later
  - `obixDriver` 3.3.28.2 or later

    The AxSupervisor also requires the "`obixDriver`" feature in its license. Any AX JACE that needs direct Obix client access to any R2 JACE 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:

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

Engineering summary

Most engineering is expected to occur in the AxSupervisor 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 JACEs is modeled under the Drivers/NiagaraNetwork, as for any AxSupervisor. 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.

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 JACEs, as well as other R2-specific topics.

AX station engineering summary

Add an ObixNetwork in the AxSupervisor'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 "AxSupervisor engineering" on page 2-7.

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:

- Reducing R2 station resource count
- Reducing R2 swid lengths

Reducing R2 station resource count

Typically, the reason for integrating an R2 JACE in an AxSupervisor is to provide access to it via AX PxPages on the AxSupervisor. 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:

```
/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...

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 an 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
```

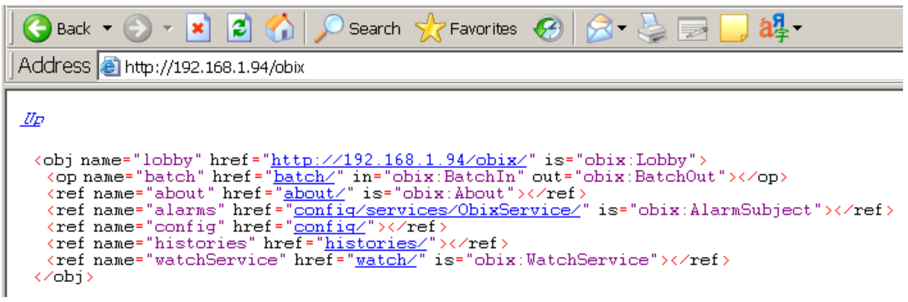
where <host> is IP address or hostname, and [:port] is optional (if omitted, assumed as 80).

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 32 *Example browser connection to confirm R2 station oBIX server operation*



```
<obj name="lobby" href="http://192.168.1.94/obix/" is="obix:Lobby">
  <op name="batch" href="batch/" in="obix:BatchIn" out="obix:BatchOut"></op>
  <ref name="about" href="about/" is="obix:About"></ref>
  <ref name="alarms" href="config/services/ObixService/" is="obix:AlarmSubject"></ref>
  <ref name="config" href="config/"></ref>
  <ref name="histories" href="histories/"></ref>
  <ref name="watchService" href="watch/" is="obix:WatchService"></ref>
</obj>
```

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 an AX station). This simplifies engineering on the R2 side.

NOTE: For this reason, the AxSupervisor 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

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 (fIn) of the export object.

Figure 33 *Example ObixAnalogExport object copied into station for linking into AnalogOutput object*

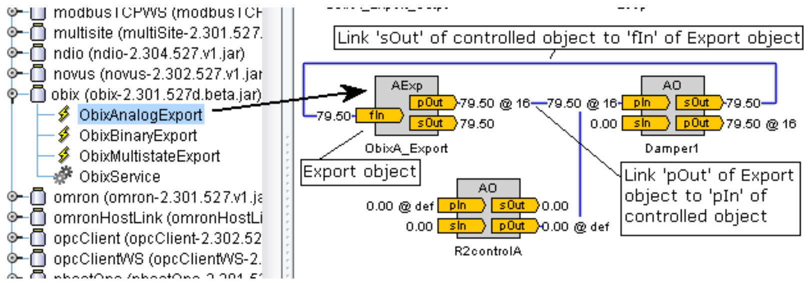

modbusTCPWS (modbusTCF
multisite (multiSite-2.301.527.
ndio (ndio-2.304.527.v1.jar)
novus (novus-2.302.527.v1.jar
obix (obix-2.301.527d.beta.jar)
ObixAnalogExport
ObixBinaryExport
ObixMultistateExport
ObixService
omron (omron-2.301.527.v1.ja
omronHostLink (omronHostLi
opcClient (opcClient-2.302.52
opcClientWS (opcClientWS-2.
Link 'sOut' of controlled object to 'fIn' of Export object
AExp
pOut
79.50 @ 16
79.50
fIn
sOut
79.50
ObixA_Export
Export object
AO
79.50 @ 16
pIn
sOut
79.50
0.00
sIn
pOut
79.50 @ 16
Damper1
Link 'pOut' of Export object to 'pIn' of controlled object
AO
0.00 @ def
pIn
sOut
0.00
0.00
sIn
pOut
0.00 @ def
R2controlA

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

Figure 34 ObixBinaryExport object copied into station for linking into BinaryOutput (BO) object

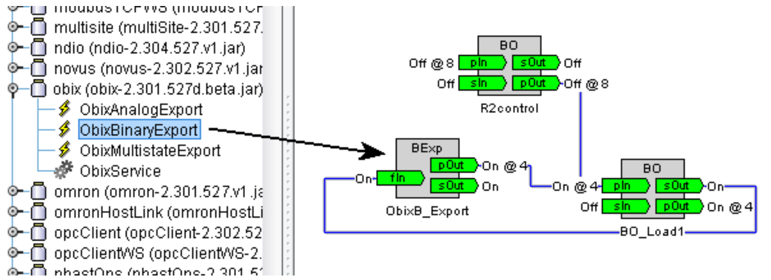

multisite (multiSite-2.301.527.
ndio (ndio-2.304.527.v1.jar)
novus (novus-2.302.527.v1.jar
obix (obix-2.301.527d.beta.jar)
ObixAnalogExport
ObixBinaryExport
ObixMultistateExport
ObixService
omron (omron-2.301.527.v1.ja
omronHostLink (omronHostLi
opcClient (opcClient-2.302.52
opcClientWS (opcClientWS-2.
BO
R2control
BExp
ObixB_Export
BO_Load1

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 35 ObixAnalogExport object copied into station for use as "statusInput" type value

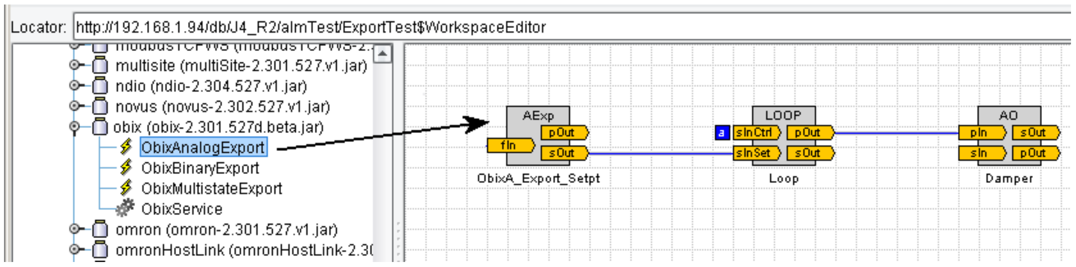

Locator: http://192.168.1.94/db/J4_R2/almTest/ExportTest$WorkspaceEditor
multisite (multiSite-2.301.527.v1.jar)
ndio (ndio-2.304.527.v1.jar)
novus (novus-2.302.527.v1.jar)
obix (obix-2.301.527d.beta.jar)
ObixAnalogExport
ObixBinaryExport
ObixMultistateExport
ObixService
omron (omron-2.301.527.v1.jar)
omronHostLink (omronHostLink-2.30
AExp
ObixA_Export_Setpt
LOOP
Loop
AO
Damper

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

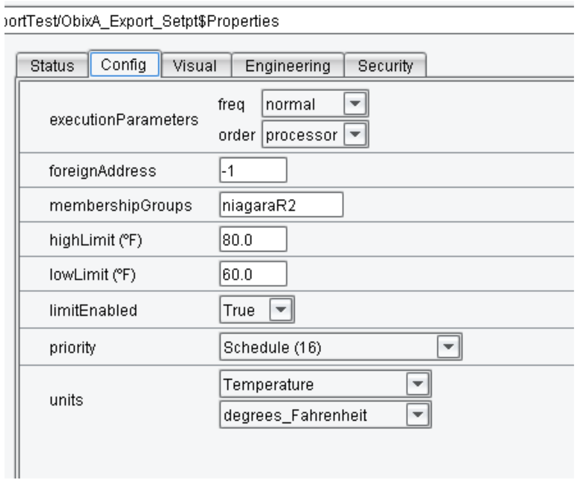
activeInactiveText (if ObixBinaryExport), or

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 36 Config properties for ObixAnalogExport, including “limit” type properties


ortTest/ObixA_Export_Setpt$Properties

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

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.

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 AxSupervisor.

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 an AxSupervisor (or another AX JACE), 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 "AxSupervisor 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 a JACE-4/5, "archive_local_no_SQL").

Other possible R2 host changes

After the oBIX integration, when connecting to an R2 host with the NiagaraAX 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 JACE, but can safely be increased to 30.

AxSupervisor engineering

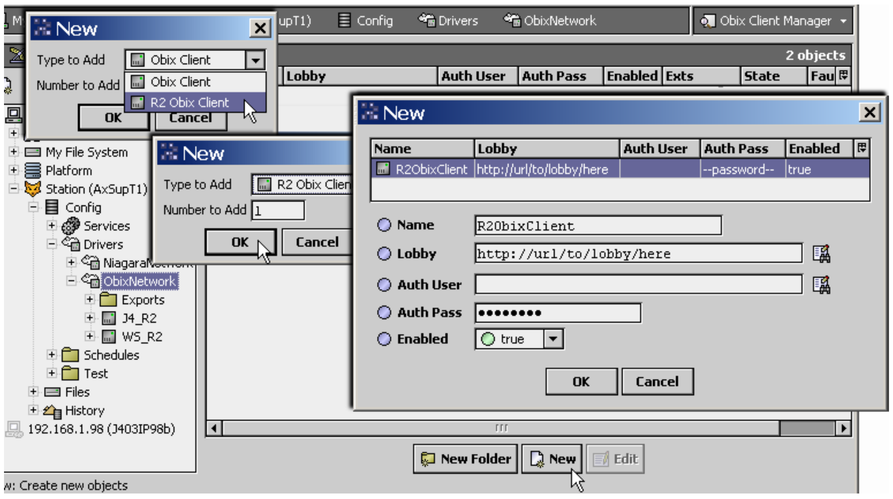
The following sections describe Niagara Supervisor engineering topics that relate to integrating R2 stations:

- ObixNetwork and R2ObixClient devices
- R2ObixClient Points
- R2ObixClient Histories (logs and archives)
- R2ObixClient Alarms
- R2ObixClient R2 Schedule exports

ObixNetwork and R2ObixClient devices

In the AxSupervisor 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 37 Add R2ObixClients using New button in Obix Client Manager view of ObixNetwork



New

Name	Lobby	Auth User	Auth Pass	Enabled
R2ObixClient	http://url/to/lobby/here		--password--	true

Name	R2ObixClient
Lobby	http://url/to/lobby/here
Auth User	
Auth Pass	••••••••
Enabled	true

OK Cancel

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 NiagaraAX 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 NiagaraAX 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 NiagaraAX 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

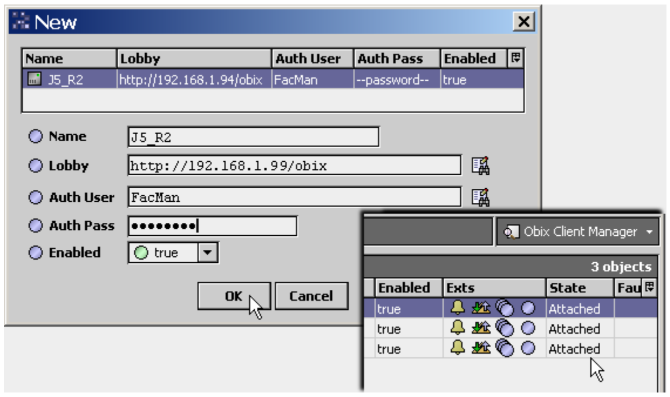
  Enter password for that user account in the R2 station.

- 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 38 Entering new ObixClient and subsequent State change to Attached



Name	Lobby	Auth User	Auth Pass	Enabled
J5_R2	http://192.168.1.94/obix	FacMan	--password--	true

Name	J5_R2
Lobby	http://192.168.1.99/obix
Auth User	FacMan
Auth Pass	••••••••
Enabled	true

3 objects

Enabled	Exts	State	Fau
true		Attached	
true		Attached	
true		Attached	

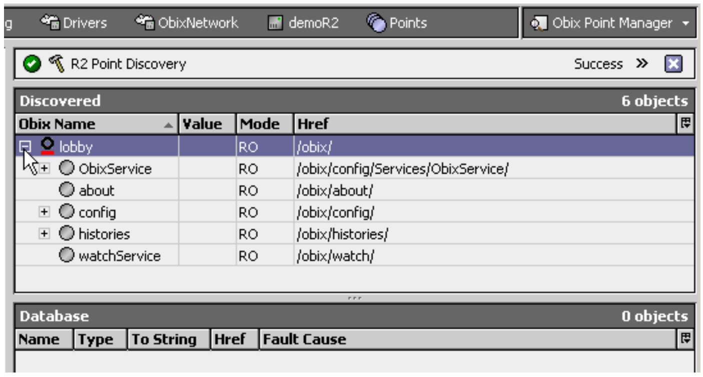
NOTE: (Troubleshooting) If after entering the R2ObixClient it remains Detached, review its Lobby syntax, including IP address of the R2 JACE (open a command prompt window and issue a `Ping` command to that IP address). Also, verify the R2 station user credentials for Auth User and Auth Pass are correct. 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.

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 39 Top-level lobby structure in R2 station



R2 Point Discovery Success

Discovered 6 objects

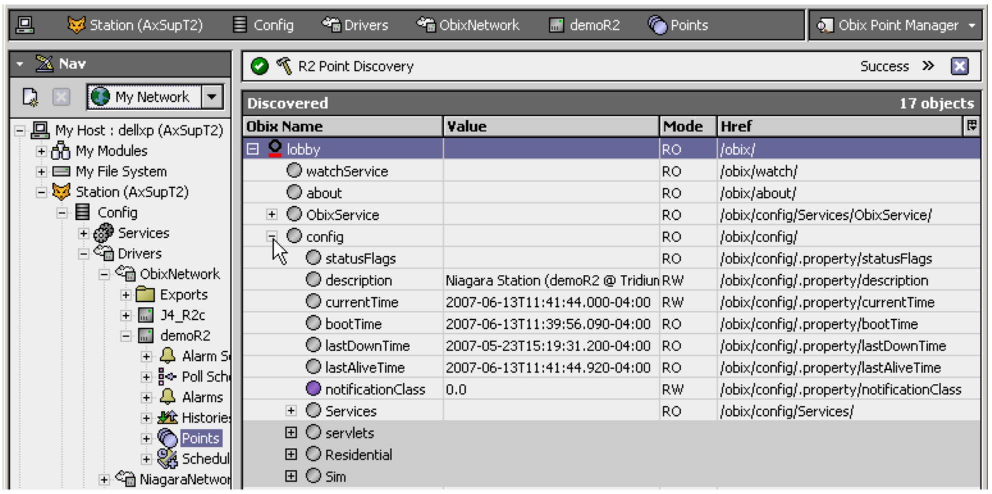
Obix Name	Value	Mode	Href
lobby		RO	/obix/
ObixService		RO	/obix/config/Services/ObixService/
about		RO	/obix/about/
config		RO	/obix/config/
histories		RO	/obix/histories/
watchService		RO	/obix/watch/

Database 0 objects

Name	Type	To String	Href	Fault Cause

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 40 ObixPointManager with lobby expanded to show discovered proxy point candidates

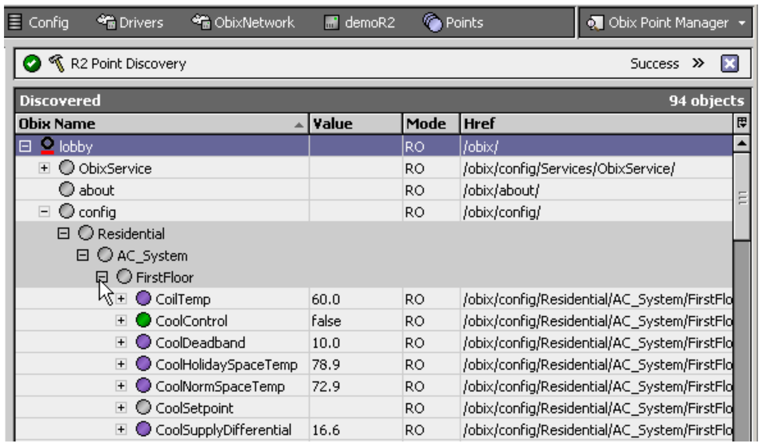


Discovered 17 objects

Obix Name	Value	Mode	Href
lobby		RO	/obix/
watchService		RO	/obix/watch/
about		RO	/obix/about/
ObixService		RO	/obix/config/Services/ObixService/
config		RO	/obix/config/
statusFlags		RO	/obix/config/.property/statusFlags
description	Niagara Station (demoR2 @ Tridiun	RW	/obix/config/.property/description
currentTime	2007-06-13T11:41:44.000-04:00	RW	/obix/config/.property/currentTime
bootTime	2007-06-13T11:39:56.090-04:00	RO	/obix/config/.property/bootTime
lastDownTime	2007-05-23T15:19:31.200-04:00	RO	/obix/config/.property/lastDownTime
lastAliveTime	2007-06-13T11:41:44.920-04:00	RO	/obix/config/.property/lastAliveTime
notificationClass	0.0	RW	/obix/config/.property/notificationClass
Services		RO	/obix/config/Services/
servlets			
Residential			
Sim			

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 41 Use column header sorts and expand containers as needed to find R2 objects and properties



Discovered 94 objects

Obix Name	Value	Mode	Href
lobby		RO	/obix/
ObixService		RO	/obix/config/Services/ObixService/
about		RO	/obix/about/
config		RO	/obix/config/
Residential			
AC_System			
FirstFloor			
CoilTemp	60.0	RO	/obix/config/Residential/AC_System/FirstFlo
CoolControl	false	RO	/obix/config/Residential/AC_System/FirstFlo
CoolDeadband	10.0	RO	/obix/config/Residential/AC_System/FirstFlo
CoolHolidaySpaceTemp	78.9	RO	/obix/config/Residential/AC_System/FirstFlo
CoolNormSpaceTemp	72.9	RO	/obix/config/Residential/AC_System/FirstFlo
CoolSetpoint		RO	/obix/config/Residential/AC_System/FirstFlo
CoolSupplyDifferential	16.6	RO	/obix/config/Residential/AC_System/FirstFlo

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:

- 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.

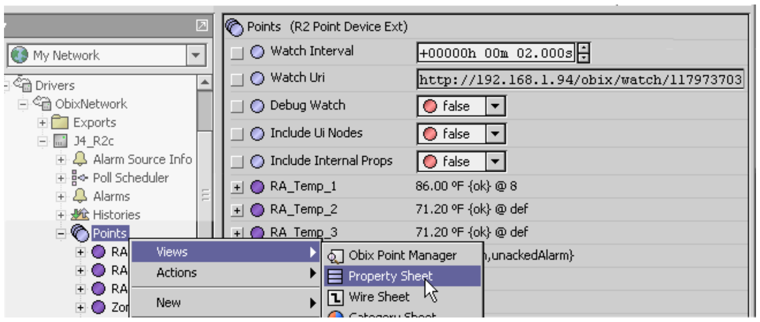
- By default, all R2 Gx objects (GxText, GxFan, and so on) are omitted from the discovered lobby tree, although GxPage containers still appear. This is controlled by the setting of the property of the Points extension of the parent R2ObixClient: **Include Ui Nodes** (default value is `false`). It is 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 42 *R2PointsDeviceExt property sheet with Include defaults*



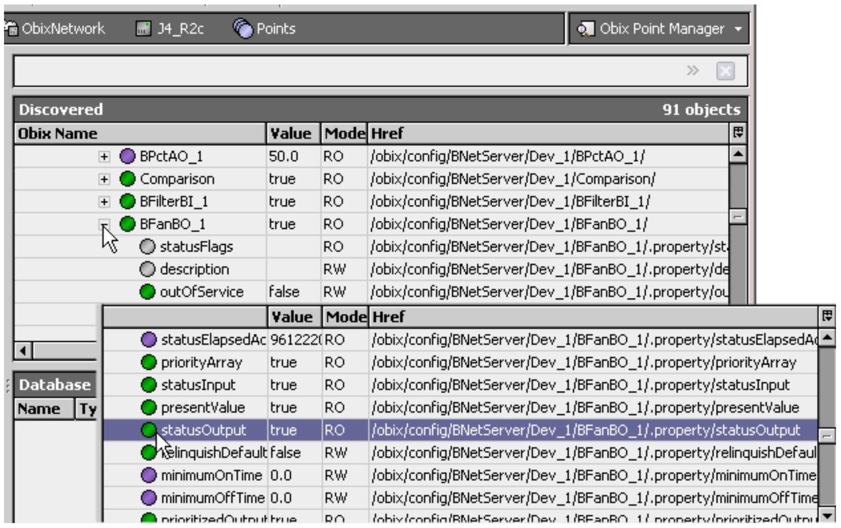
Points (R2 Point Device Ext)

Watch Interval	+00000h 00m 02.000s
Watch Uri	http://192.168.1.94/obix/watch/117973703
Debug Watch	false
Include Ui Nodes	false
Include Internal Props	false
RA_Temp_1	86.00 °F {ok} @ 8
RA_Temp_2	71.20 °F {ok} @ def
RA_Temp_3	71.20 °F {ok} @ def
	,unackedAlarm}

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 43 *Select property of a discovered R2 object, vs. root (Node) object*



Discovered 91 objects

Obix Name	Value	Mode	Href
BPctAO_1	50.0	RO	/obix/config/BNetServer/Dev_1/BPctAO_1/
Comparison	true	RO	/obix/config/BNetServer/Dev_1/Comparison/
BFilterBI_1	true	RO	/obix/config/BNetServer/Dev_1/BFilterBI_1/
BFanBO_1	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/
statusFlags		RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/st
description		RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/de
outOfService	false	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/ou

	Value	Mode	Href
statusElapsedAc	961222(	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusElapsedAc
priorityArray	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/priorityArray
statusInput	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusInput
presentValue	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/presentValue
statusOutput	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusOutput
relinquishDefault	false	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/relinquishDefaul
minimumOnTime	0.0	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/minimumOnTime
minimumOffTime	0.0	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/minimumOffTime
prioritizedOutput	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/prioritizedOutpu

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.

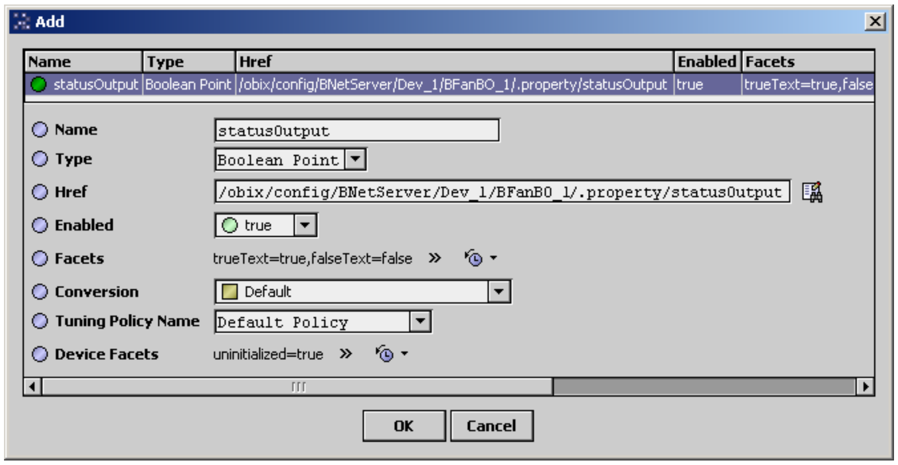
In summary, selecting object properties (rather than their root objects) will have a big impact on performance of the oBIX integration. Typically, an order of magnitude or two can be gained by specifying a property for each proxy point, versus specifying the root object.

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

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 44 *Example Add dialog for R2 Obix proxy point*



Add

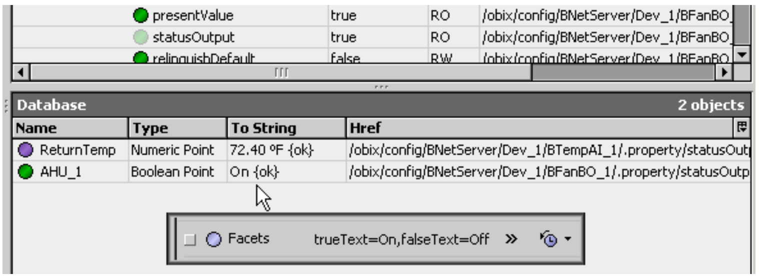
Name	Type	Href	Enabled	Facets
statusOutput	Boolean Point	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusOutput	true	trueText=true,false

Name	statusOutput
Type	Boolean Point
Href	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusOutput
Enabled	true
Facets	trueText=true,falseText=false
Conversion	Default
Tuning Policy Name	Default Policy
Device Facets	uninitialized=true

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 45 *Facets are automatically learned from the R2 object's units or activeInactiveText*



presentValue	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_
statusOutput	true	RO	/obix/config/BNetServer/Dev_1/BFanBO_
relinquishDefault	false	RW	/obix/config/BNetServer/Dev_1/BFanBO_

Database 2 objects

Name	Type	To String	Href
ReturnTemp	Numeric Point	72.40 ºF {ok}	/obix/config/BNetServer/Dev_1/BTempAI_1/.property/statusOut
AHU_1	Boolean Point	On {ok}	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusOutp

Facets trueText=On,falseText=Off

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

About Obix proxy points

Added Obix proxy points often resemble Niagara proxy points (Niagaraunder Points container of a Niagara Station in a NiagaraNetwork) 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

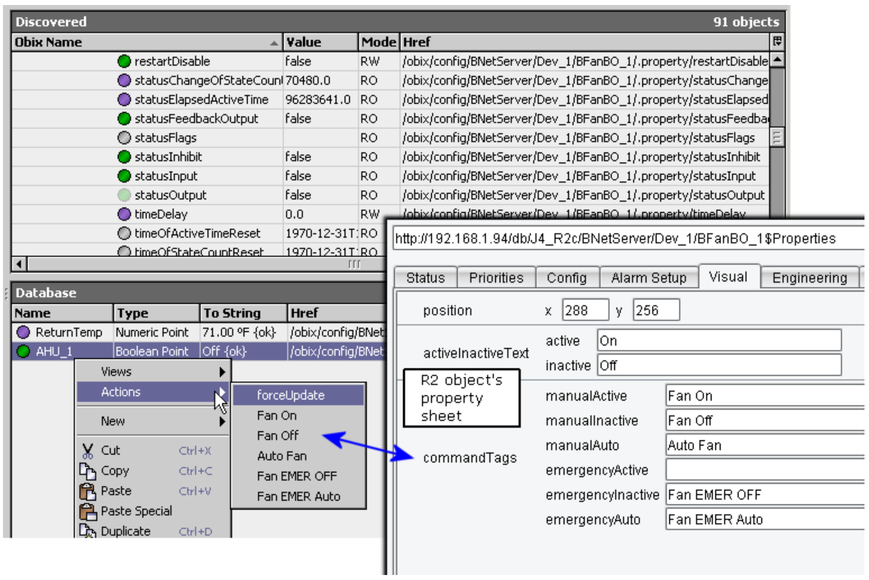
  native AX actions for the writable point - to change the value of the specific R2 property

  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 46 *Proxy point for R2 object or property offers actions for commands, even as read-only point*



Discovered 91 objects

Obix Name	Value	Mode	Href
restartDisable	false	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/restartDisable
statusChangeOfStateCoun	70480.0	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusChange
statusElapsedActiveTime	96283641.0	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusElapsed
statusFeedbackOutput	false	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusFeedba
statusFlags		RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusFlags
statusInhibit	false	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusInhibit
statusInput	false	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusInput
statusOutput	false	RO	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/statusOutput
timeDelay	0.0	RW	/obix/config/BNetServer/Dev_1/BFanBO_1/.property/timeDelay
timeOfActiveTimeReset	1970-12-31T	RO	
timeOfStateCountReset	1970-12-31T	RO	

Database

Name	Type	To String	Href
ReturnTemp	Numeric Point	71.00 ºF {ok}	/obix/config/BNet
AHU_1	Boolean Point	Off {ok}	/obix/config/BNet

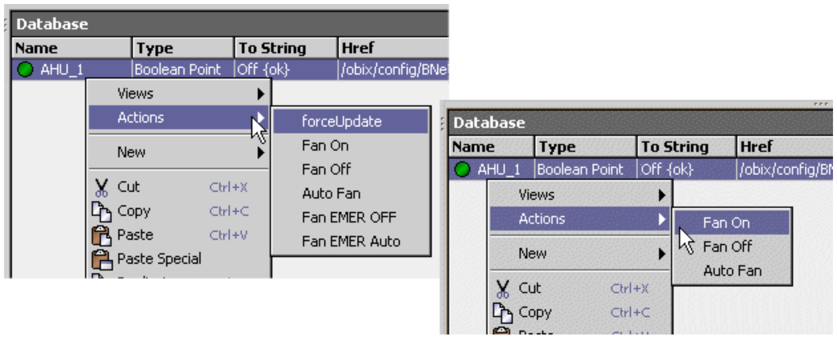
http://192.168.1.94/db/J4_R2c/BNetServer/Dev_1/BFanBO_1$Properties

position	x	288
	y	256
activeInactiveText	active	On
	inactive	Off
commandTags	manualActive	Fan On
	manualInactive	Fan Off
	manualAuto	Auto Fan
	emergencyActive	
	emergencyInactive	Fan EMER OFF
	emergencyAuto	Fan EMER Auto

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 47 *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).

Command Notes

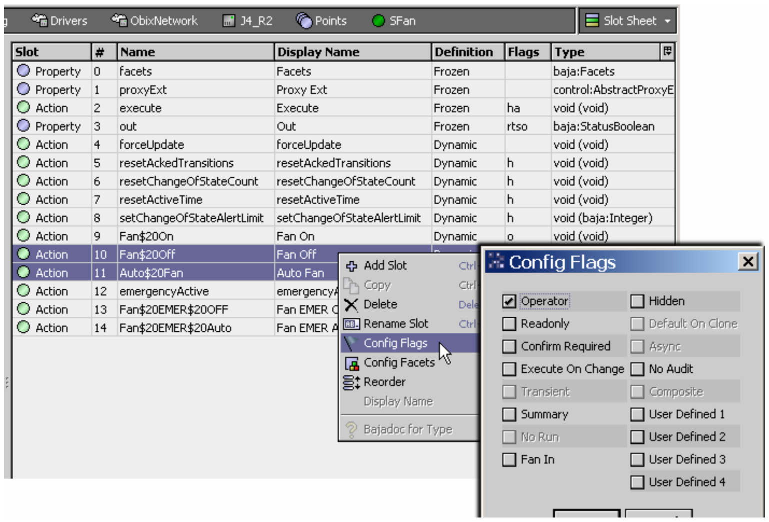
In addition to a proxy point "forceUpdate" action, note by default "normal right-click" commands appear on the action menu for Obix proxy points—providing the source R2 object has commands. Actions display mirroring the configured R2 object's "commandTags" property strings, where applicable.

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 48 *Obix proxy point from its slot sheet, showing Operator flag set on manual-level actions*



Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	facets	Facets	Frozen		baja:Facets
Property	1	proxyExt	Proxy Ext	Frozen		control:AbstractProxyE
Action	2	execute	Execute	Frozen	ha	void (void)
Property	3	out	Out	Frozen	rtso	baja:StatusBoolean
Action	4	forceUpdate	forceUpdate	Dynamic		void (void)
Action	5	resetAckedTransitions	resetAckedTransitions	Dynamic	h	void (void)
Action	6	resetChangeOfStateCount	resetChangeOfStateCount	Dynamic	h	void (void)
Action	7	resetActiveTime	resetActiveTime	Dynamic	h	void (void)
Action	8	setChangeOfStateAlertLimit	setChangeOfStateAlertLimit	Dynamic	h	void (baja:Integer)
Action	9	Fan$20On	Fan On	Dynamic	o	void (void)
Action	10	Fan$20Off	Fan Off			
Action	11	Auto$20Fan	Auto Fan			
Action	12	emergencyActive	emergencyA			
Action	13	Fan$20EMER$20OFF	Fan EMER C			
Action	14	Fan$20EMER$20Auto	Fan EMER A			

[✓] Operator	Hidden
Readonly	Default On Clone
Confirm Required	Async
Execute On Change	No Audit
Transient	Composite
Summary	User Defined 1
No Run	User Defined 2
Fan In	User Defined 3
	User Defined 4

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.

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.

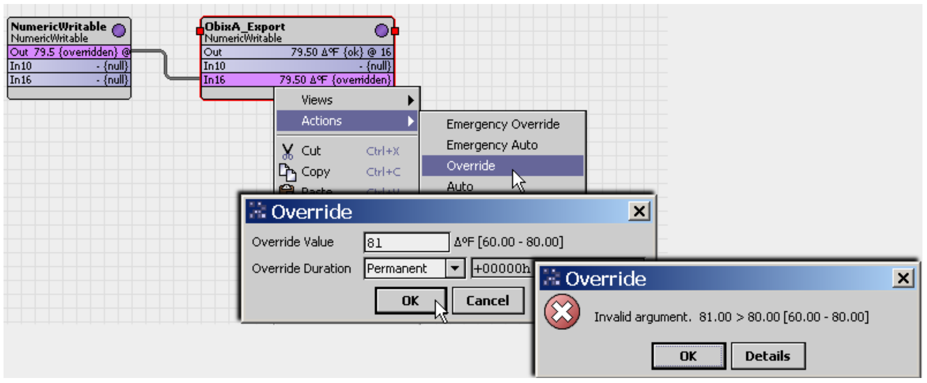
- In a remote AX station, you do this by making a reciprocal Niagara proxy point looking back at the “controlling” (output) point in the local station. See “Link control and Niagara proxy points” in the *NiagaraAX Drivers Guide* for details.
- In the case of the R2 station, you copy one of the three 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 (fIn) 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.

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.



Override

Override Value 81 Δ°F [60.00 - 80.00]

Override Duration Permanent +00000h

Override

Invalid argument. 81.00 > 80.00 [60.00 - 80.00]

Figure 49 *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"/>
```

ObixClient proxy point tips

When creating Obix proxy points for an R2 JACE 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 an 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.

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:

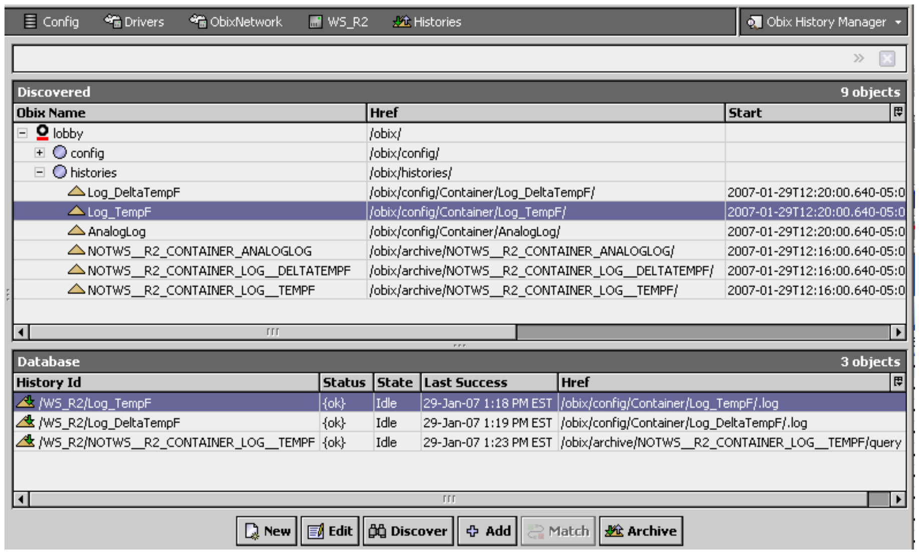
```
HTTP Error 503:Service Unavailable
```

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

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 50 *ObixHistoryManager is for importing R2 logs and/or archives into the history space*



Discovered 9 objects

Obix Name	Href	Start
lobby	/obix/	
config	/obix/config/	
histories	/obix/histories/	
Log_DeltaTempF	/obix/config/Container/Log_DeltaTempF/	2007-01-29T12:20:00.640-05:0
Log_TempF	/obix/config/Container/Log_TempF/	2007-01-29T12:20:00.640-05:0
AnalogLog	/obix/config/Container/AnalogLog/	2007-01-29T12:20:00.640-05:0
NOTWS__R2_CONTAINER_ANALOGLOG	/obix/archive/NOTWS__R2_CONTAINER_ANALOGLOG/	2007-01-29T12:16:00.640-05:0
NOTWS__R2_CONTAINER_LOG__DELTATEMPF	/obix/archive/NOTWS__R2_CONTAINER_LOG__DELTATEMPF/	2007-01-29T12:16:00.640-05:0
NOTWS__R2_CONTAINER_LOG__TEMPF	/obix/archive/NOTWS__R2_CONTAINER_LOG__TEMPF/	2007-01-29T12:16:00.640-05:0

Database 3 objects

History Id	Status	State	Last Success	Href
/WS_R2/Log_TempF	{ok}	Idle	29-Jan-07 1:18 PM EST	/obix/config/Container/Log_TempF/.log
/WS_R2/Log_DeltaTempF	{ok}	Idle	29-Jan-07 1:19 PM EST	/obix/config/Container/Log_DeltaTempF/.log
/WS_R2/NOTWS__R2_CONTAINER_LOG__TEMPF	{ok}	Idle	29-Jan-07 1:23 PM EST	/obix/archive/NOTWS__R2_CONTAINER_LOG__TEMPF/query

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

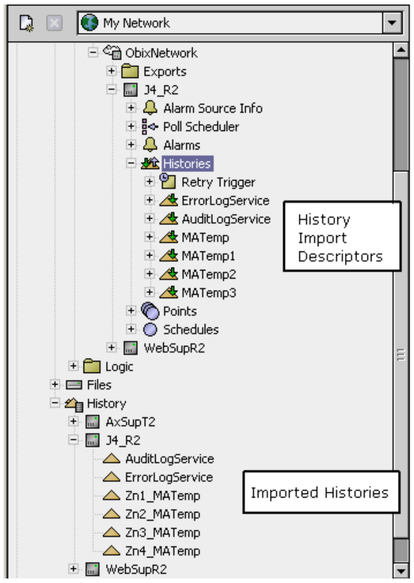
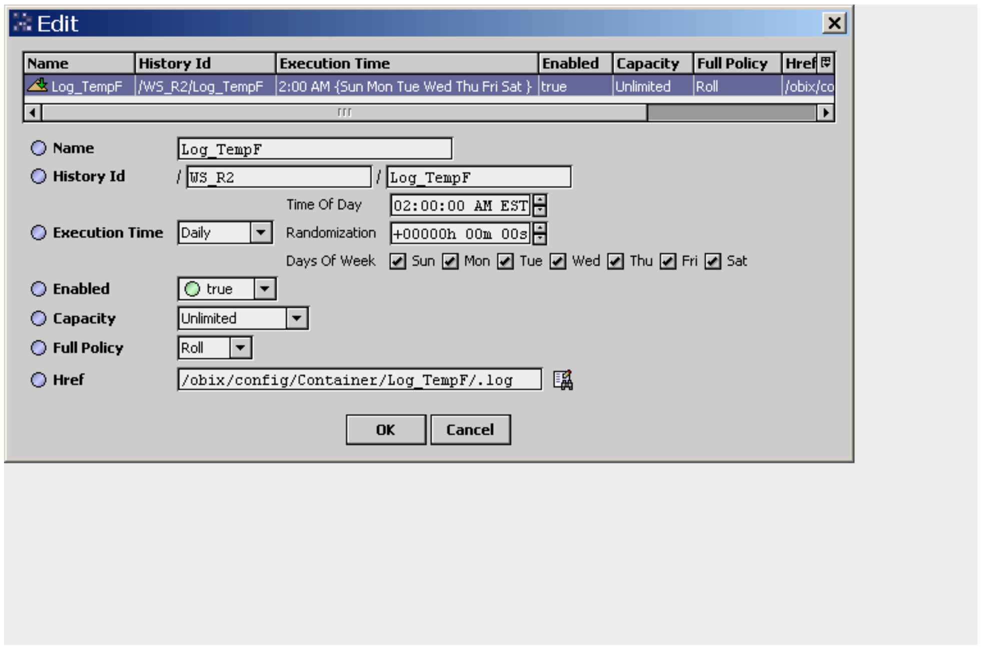


Figure 2-22 shows an example add/edit dialog for an ObixHistoryImport descriptor with default values, where the import descriptor name matches the source R2 Log object name, and the history ID is a combination of the R2ObixClient's name / Log object name.

Figure 52 *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 an 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.

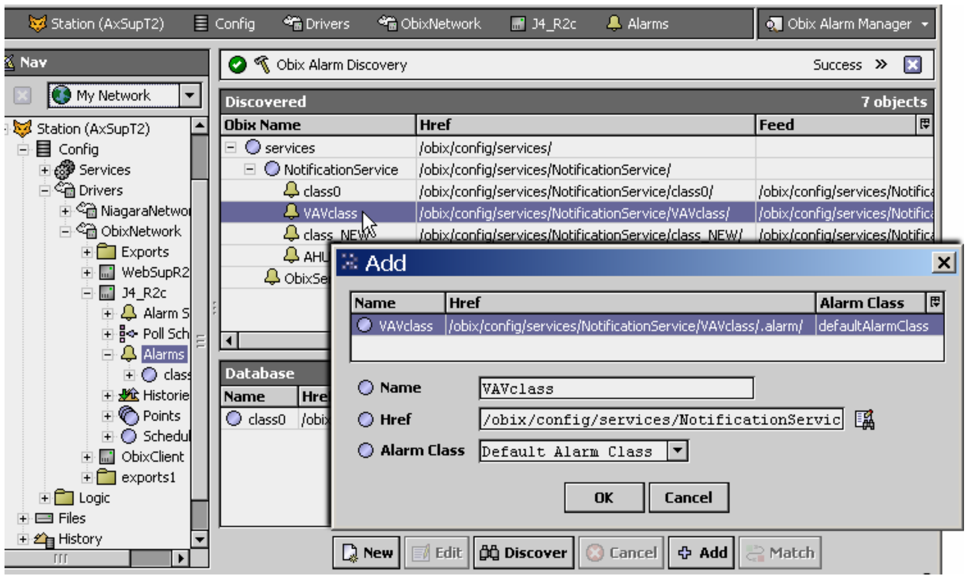
  Although the Obix History Import manager allows you to create multiple import descriptors (with default values) for an identical Obix Name, note that only one can successfully import using the same History Id. Import descriptors with a duplicate History Id will go into fault upon import attempt. Therefore, 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.

R2ObixClient Alarms

The Alarms extension of an R2ObixClient for an 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 53 Obix Alarm Manager view used to add alarm feeds from R2 NotificationClass objects



Obix Alarm Discovery Success

Discovered 7 objects

Obix Name	Href	Feed
services	/obix/config/services/	
NotificationService	/obix/config/services/NotificationService/	
class0	/obix/config/services/NotificationService/class0/	/obix/config/services/Notifica
VAVclass	/obix/config/services/NotificationService/VAVclass/	/obix/config/services/Notifica
class_NEW	/obix/config/services/NotificationService/class_NEW/	/obix/config/services/Notifica
AHU		
ObixSe		

Database

Name	Hre
class0	/obix

Add

Name	Href	Alarm Class
VAVclass	/obix/config/services/NotificationService/VAVclass/.alarm/	defaultAlarmClass

Name	VAVclass
Href	/obix/config/services/NotificationServic
Alarm Class	Default Alarm Class

OK Cancel

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:

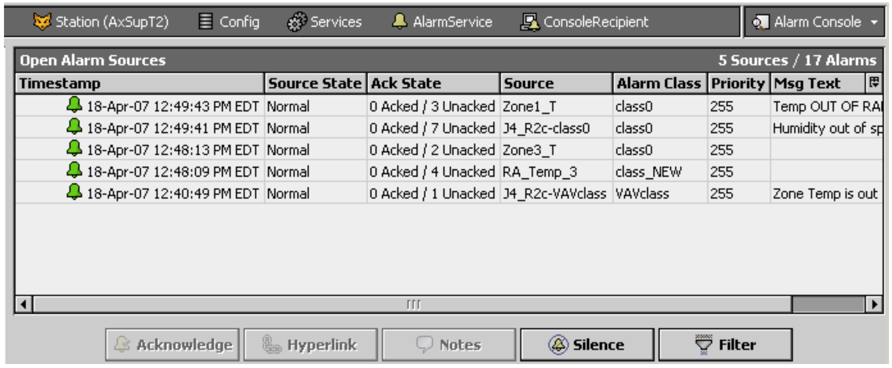
- R2 alarm import operation
- Example R2 alarm imports
- Final notes on imported R2 alarms

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 NiagaraAX alarming subsystem). However, R2 alarms do support the oBIX “AckAlarm” contract, allowing acknowledgment from NiagaraAX.

Thus, as shown below, all R2 native alarms appear as “normal” (green alarm bell) source state events when routed to an 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 54 *Imported R2 alarms always have “Normal” source state in 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 sp
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

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 an 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.

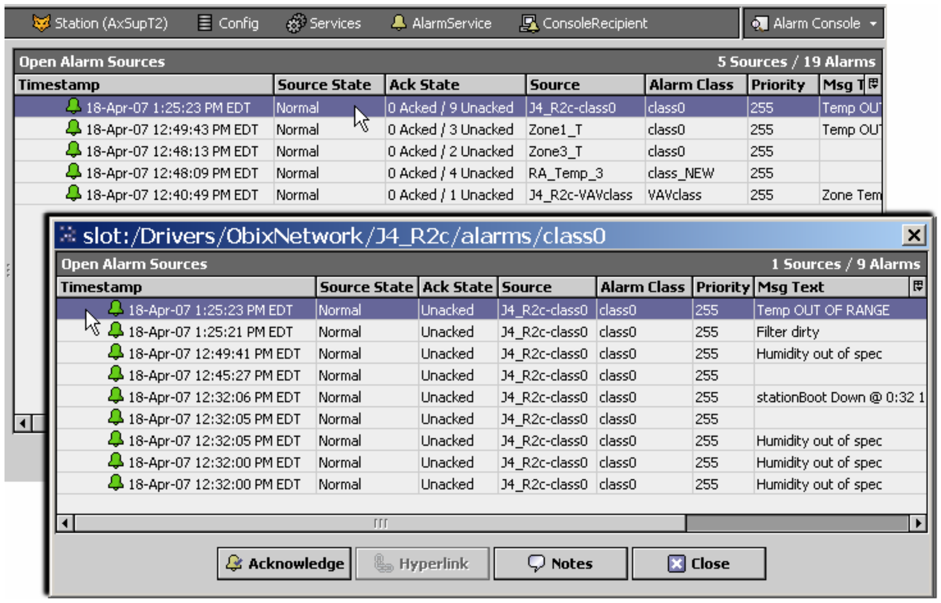
NOTE: In this case especially, you must be careful about acknowledgement of a single row in the Alarm Console, as it may represent several different R2 alarm sources—the single acknowledgement will be applied to all underlying alarms (whether you are aware of them or not).

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

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 55 *R2 alarms by objects not mapped as proxy points require Alarm Details inspection*



Open Alarm Sources — 5 Sources / 19 Alarms

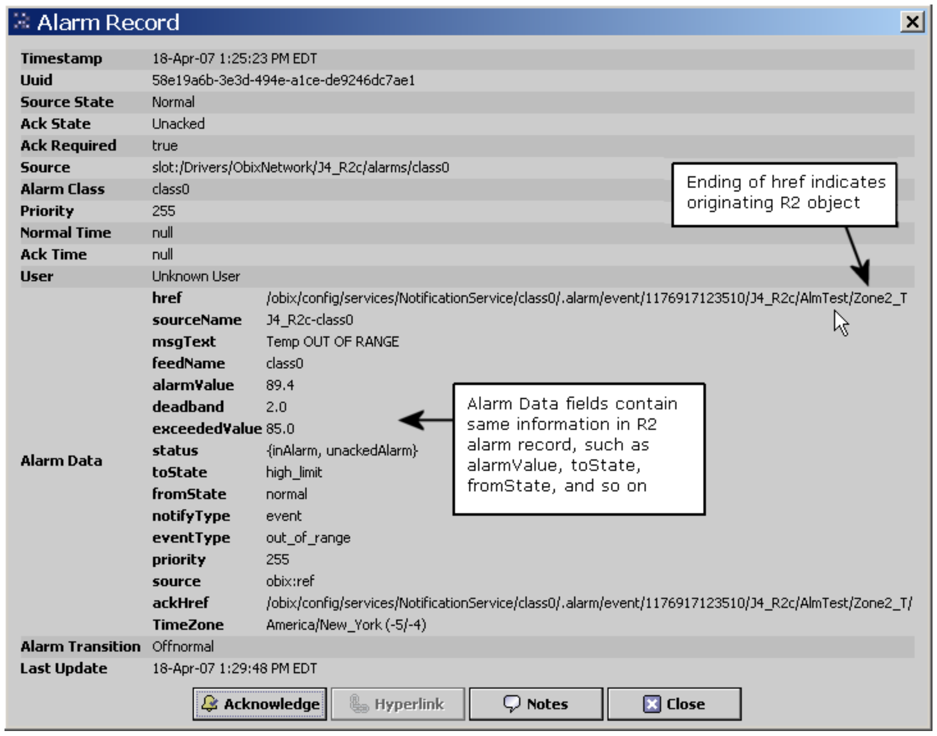
Timestamp	Source State	Ack State	Source	Alarm Class	Priority	Msg T
18-Apr-07 1:25:23 PM EDT	Normal	0 Acked / 9 Unacked	J4_R2c-class0	class0	255	Temp OU
18-Apr-07 12:49:43 PM EDT	Normal	0 Acked / 3 Unacked	Zone1_T	class0	255	Temp OU
18-Apr-07 12:48:13 PM EDT	Normal	0 Acked / 2 Unacked	Zone3_T	class0	255	
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 Tem

slot:/Drivers/ObixNetwork/J4_R2c/alarms/class0

Open Alarm Sources — 1 Sources / 9 Alarms

Timestamp	Source State	Ack State	Source	Alarm Class	Priority	Msg Text
18-Apr-07 1:25:23 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Temp OUT OF RANGE
18-Apr-07 1:25:21 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Filter dirty
18-Apr-07 12:49:41 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Humidity out of spec
18-Apr-07 12:45:27 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	
18-Apr-07 12:32:06 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	stationBoot Down @ 0:32 1
18-Apr-07 12:32:05 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	
18-Apr-07 12:32:05 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Humidity out of spec
18-Apr-07 12:32:00 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Humidity out of spec
18-Apr-07 12:32:00 PM EDT	Normal	Unacked	J4_R2c-class0	class0	255	Humidity out of spec

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


Alarm Record
Ending of href indicates originating R2 object
Alarm Data fields contain same information in R2 alarm record, such as alarmValue, toState, fromState, and so on
Acknowledge
Hyperlink
Notes
Close

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	

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.

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 an 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 (JACE) 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

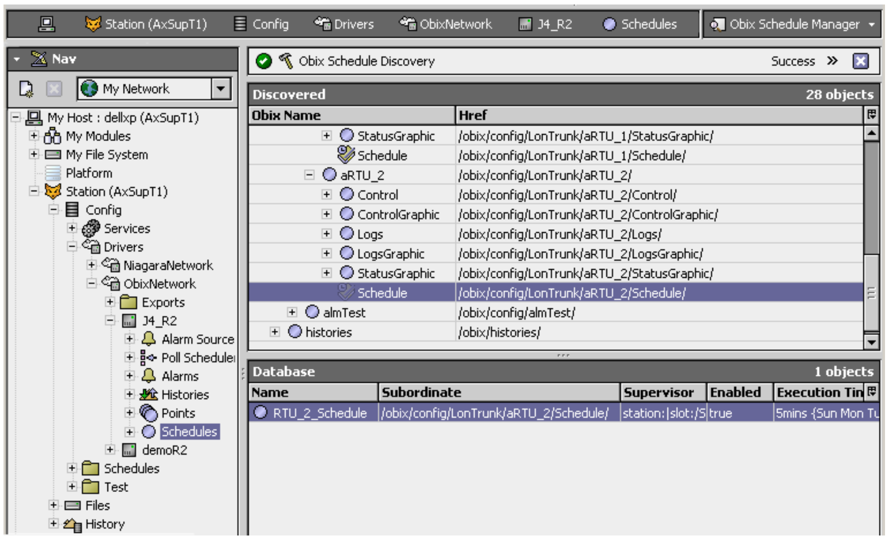
If alarming is critical, for the reasons previously noted you may wish to transition all alarming to "native AX" alarming, instead of using the ObixClient Alarms feature. Do this by creating Obix proxy points for all objects that require alarming (or runtime/COS count events), and then adding to each 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.

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 56 Obix Schedule Manager view of Schedules device extension to designate R2 slave schedules



Discovered 28 objects

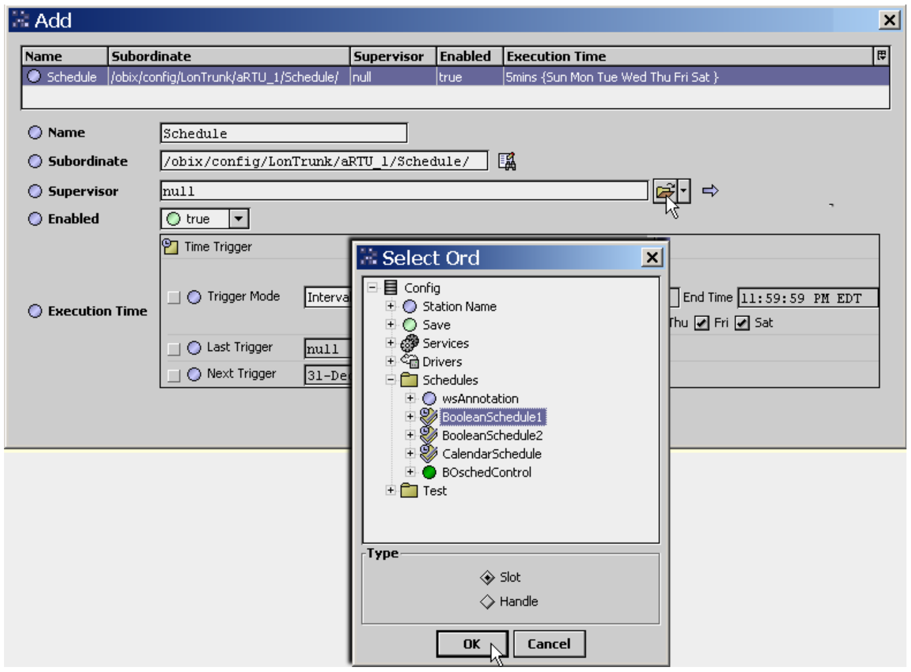
Obix Name	Href
StatusGraphic	/obix/config/LonTrunk/aRTU_1/StatusGraphic/
Schedule	/obix/config/LonTrunk/aRTU_1/Schedule/
aRTU_2	/obix/config/LonTrunk/aRTU_2/
Control	/obix/config/LonTrunk/aRTU_2/Control/
ControlGraphic	/obix/config/LonTrunk/aRTU_2/ControlGraphic/
Logs	/obix/config/LonTrunk/aRTU_2/Logs/
LogsGraphic	/obix/config/LonTrunk/aRTU_2/LogsGraphic/
StatusGraphic	/obix/config/LonTrunk/aRTU_2/StatusGraphic/
Schedule	/obix/config/LonTrunk/aRTU_2/Schedule/
almTest	/obix/config/almTest/
histories	/obix/histories/

Database 1 objects

Name	Subordinate	Supervisor	Enabled	Execution Tin
RTU_2_Schedule	/obix/config/LonTrunk/aRTU_2/Schedule/	station:|slot:/S	true	5mins {Sun Mon Tu

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 57 Add/Edit dialog for ObixScheduleExport descriptor, showing Supervisor



Name	Subordinate	Supervisor	Enabled	Execution Time
Schedule	/obix/config/LonTrunk/aRTU_1/Schedule/	null	true	5mins {Sun Mon Tue Wed Thu Fri Sat }

Name	Schedule
Subordinate	/obix/config/LonTrunk/aRTU_1/Schedule/
Supervisor	null
Enabled	true
Execution Time	Time Trigger; Trigger Mode: Interva; End Time 11:59:59 PM EDT; Thu ✓ Fri ✓ Sat; Last Trigger: null; Next Trigger: 31-De

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 an AX BooleanSchedule.

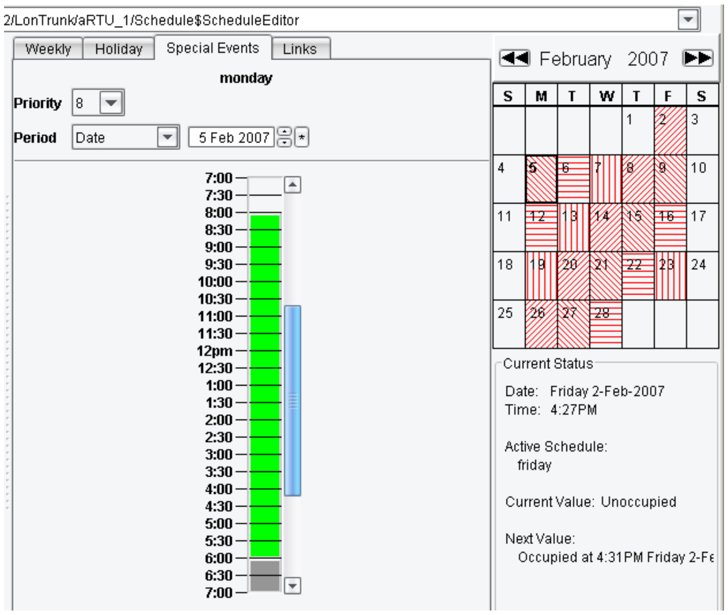
NOTE: Any target R2 Schedule should not already be "slaved" to an R2 Schedule in another station (typically in the R2 Web Supervisor station), otherwise the AX schedule supervisor function may not be successful. To remove a Schedule from R2 slaved control, delete the externalSubscription link on its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

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 58 *R2 JDE Scheduler Editor view, Special Events tab, showing received AX schedule events*


2/LonTrunk/aRTU_1/Schedule$ScheduleEditor

Priority: 8
Period: Date 5 Feb 2007

Current Status
Date: Friday 2-Feb-2007
Time: 4:27PM
Active Schedule: friday
Current Value: Unoccupied
Next Value: Occupied at 4:31PM Friday 2-Fe

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 AxSupervisor.

Chapter 4 Sample Reports Using BQL and Bound Tables

Topics covered in this chapter

♦ Creating The Report Px Page
♦ Example Reports

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.

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.

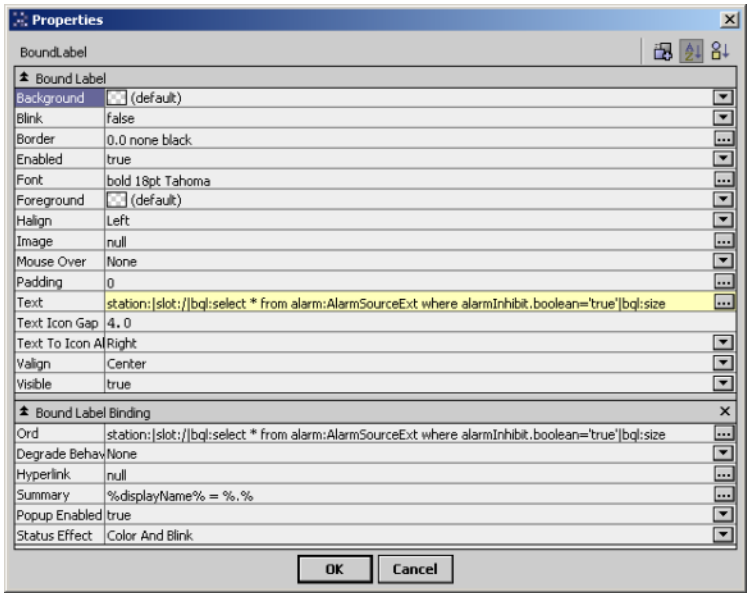
  NOTE: It is important not to put the report pane inside of a scroll pane or the auto sizing and page generation will not function correctly.

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 59 *Bound label property sheet using bql query in the Ord field*



Properties

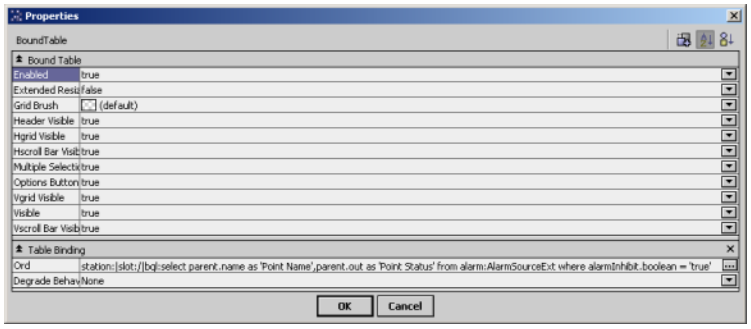
BoundLabel

Bound Label	
Background	(default)
Blink	false
Border	0.0 none black
Enabled	true
Font	bold 18pt Tahoma
Foreground	(default)
Halign	Left
Image	null
Mouse Over	None
Padding	0
Text	station:|slot:/|bql:select * from alarm:AlarmSourceExt where alarmInhibit.boolean='true'|bql:size
Text Icon Gap	4.0
Text To Icon Al	Right
Valign	Center
Visible	true

Bound Label Binding	
Ord	station:|slot:/|bql:select * from alarm:AlarmSourceExt where alarmInhibit.boolean='true'|bql:size
Degrade Behav	None
Hyperlink	null
Summary	%displayName% = %.%
Popup Enabled	true
Status Effect	Color And Blink

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 60 *Bound table property sheet using bql query in Ord field*



Properties

BoundTable

Bound Table	
Enabled	true
Extended Resiz	false
Grid Brush	(default)
Header Visible	true
Hgrid Visible	true
Hscroll Bar Visit	true
Multiple Selecti	true
Options Button	true
Vgrid Visible	true
Visible	true
Vscroll Bar Visib	true

Table Binding	
Ord	station:|slot:/|bql:select parent.name as 'Point Name',parent.out as 'Point Status' from alarm:AlarmSourceExt where alarmInhibit.boolean = 'true'
Degrade Behav	None

Example Reports

The following sections provide examples of how to create specific types of reports. Report examples may be available in both the Px and PDF file formats from Niagara-Central.com or other Tridium support sites.

- 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.

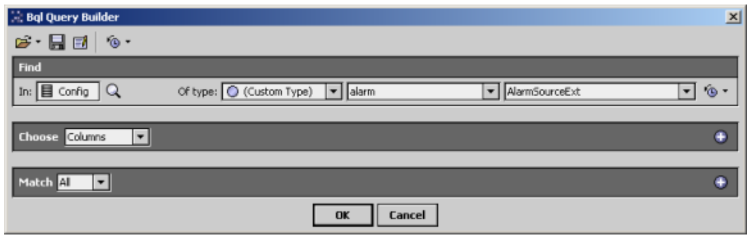
- Displaying Points with Alarm Functionality Disabled

  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 61 *Bql query builder constructing query for alarm source extensions*

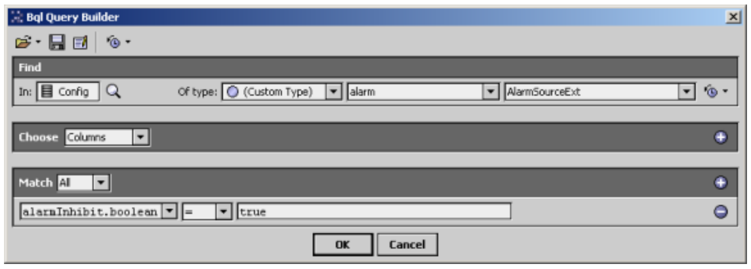


– Filtering by Alarm Inhibit Property

  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 62 *Bql query builder constructing query filtered by alarm inhibit property*

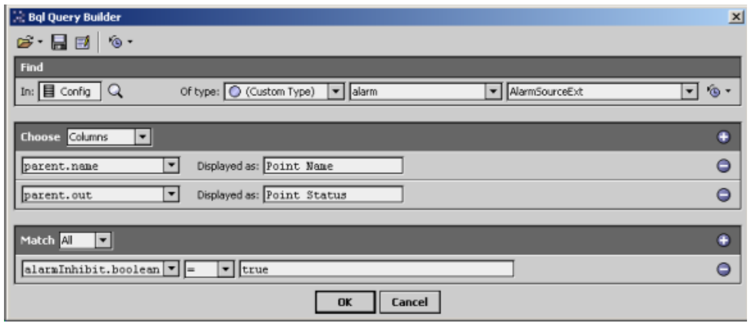


– Displaying Specific Columns

  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.boolean = 'true'
```

Figure 63 *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 'lbql: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.

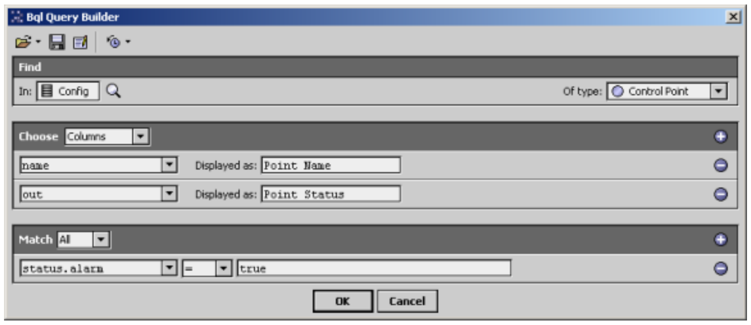
- Displaying Points Currently In Alarm

  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 64 *Bql query builder constructing query for control points in alarm*

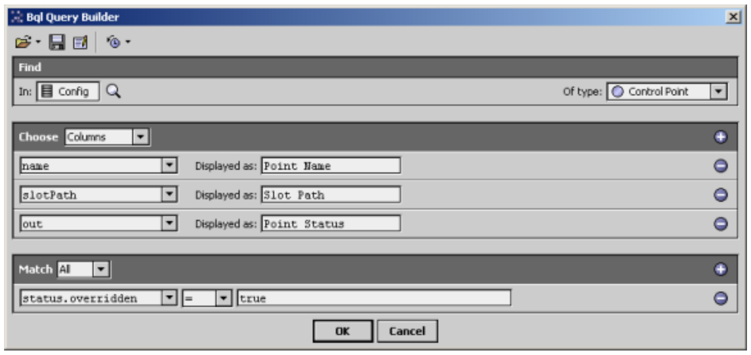


- Displaying Points Currently Overridden

  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 65 Bql query builder constructing query for control points currently overridden



Bql Query Builder

Find

In: Config Of type: Control Point

Choose Columns

name	Displayed as: Point Name
slotPath	Displayed as: Slot Path
out	Displayed as: Point Status

Match All

status.overridden = true

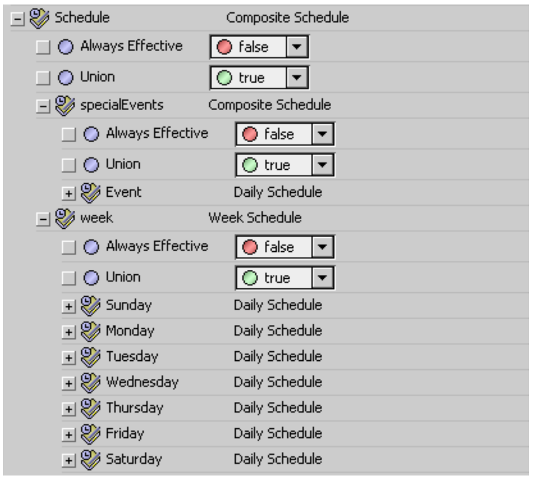
Schedule Report

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 66 Composite schedule property sheet



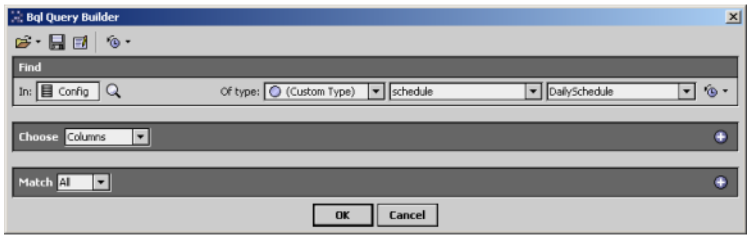
Schedule	Composite Schedule
Always Effective	false
Union	true
specialEvents	Composite Schedule
Always Effective	false
Union	true
Event	Daily Schedule
week	Week Schedule
Always Effective	false
Union	true
Sunday	Daily Schedule
Monday	Daily Schedule
Tuesday	Daily Schedule
Wednesday	Daily Schedule
Thursday	Daily Schedule
Friday	Daily Schedule
Saturday	Daily Schedule

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


Bql Query Builder

Find

In: Config Of type: (Custom Type) schedule DailySchedule

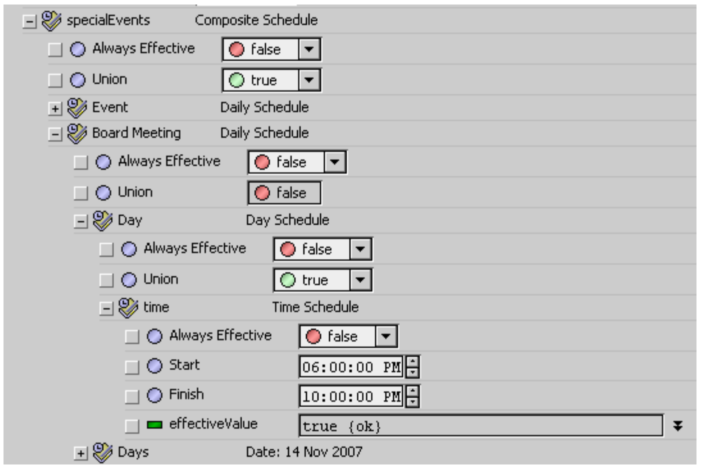
Choose Columns

Match All

OK Cancel

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


```
specialEvents          Composite Schedule
  Always Effective     false
  Union                true
  Event                Daily Schedule
  Board Meeting        Daily Schedule
    Always Effective   false
    Union              false
    Day                Day Schedule
      Always Effective false
      Union            true
      time             Time Schedule
        Always Effective false
        Start          06:00:00 PM
        Finish         10:00:00 PM
        effectiveValue true {ok}
    Days               Date: 14 Nov 2007
```

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.finsih (displays the time of the second stop event)
```

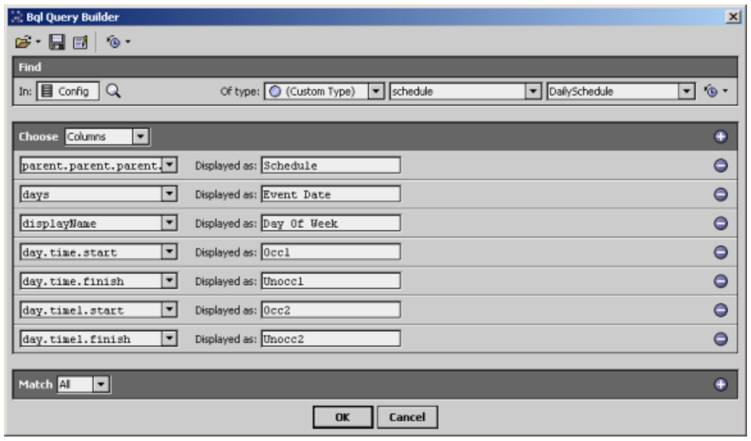
- Filtered Query for Daily Schedules

  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',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
```

  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 67 Bql query builder creating query for daily schedule components



In: Config

Of type: (Custom Type) schedule DailySchedule

Choose Columns

parent.parent.parent.	Displayed as: Schedule
days	Displayed as: Event Date
displayName	Displayed as: Day Of Week
day.time.start	Displayed as: Occ1
day.time.finish	Displayed as: Unocc1
day.time1.start	Displayed as: Occ2
day.time1.finish	Displayed as: Unocc2

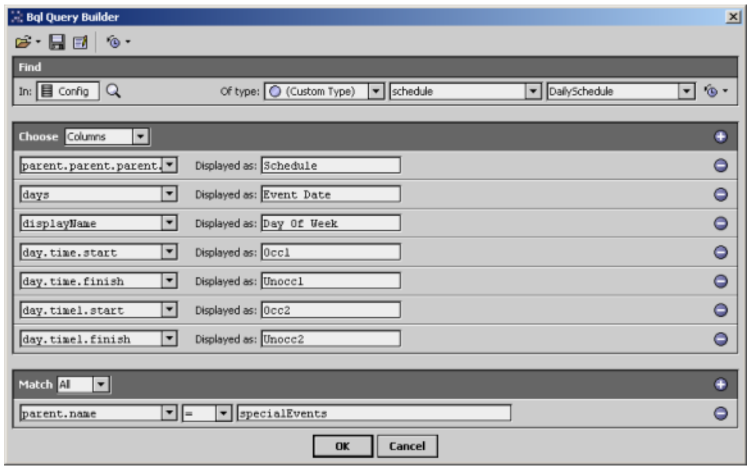
Match All

- 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 68 *Figure 12: bql query builder creating query for special events*



Bql Query Builder

Find

In: Config   Of type: (Custom Type) | schedule | DailySchedule

Choose Columns

parent.parent.parent.	Displayed as: Schedule
days	Displayed as: Event Date
displayName	Displayed as: Day Of Week
day.time.start	Displayed as: Occ1
day.time.finish	Displayed as: Unocc1
day.time1.start	Displayed as: Occ2
day.time1.finish	Displayed as: Unocc2

Match All

parent.name = specialEvents

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.

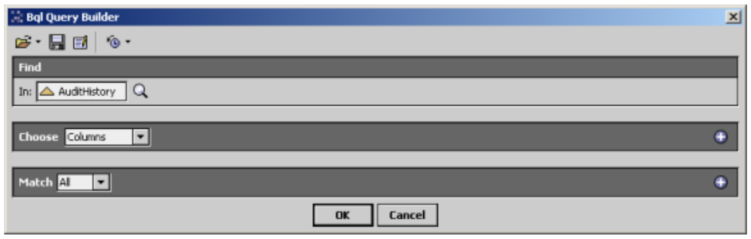
- Using the Audit Log

  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 69 *Bql query builder creating query for the audit log*



Bql Query Builder

Find

In: AuditHistory

Choose Columns

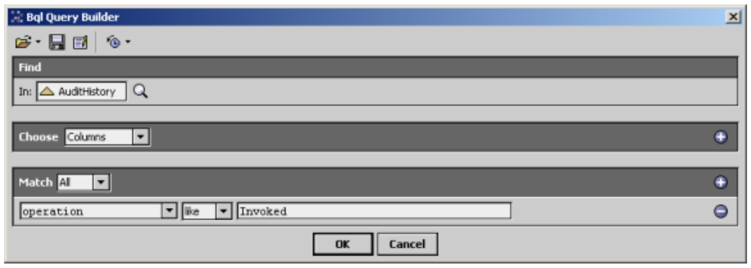
Match All

- 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 70 Figure 14: bql query builder creating query for the audit log filtered by operation



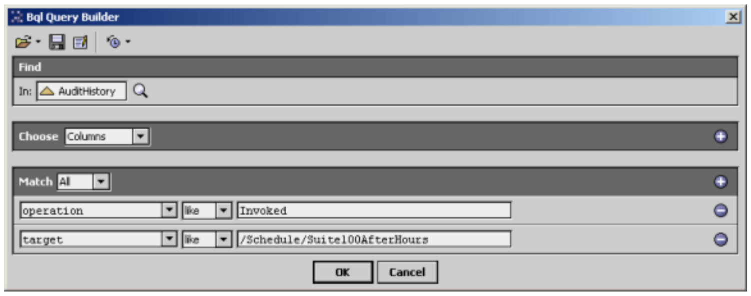
operation like Invoked

- 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 71 Bql query builder creating query for the audit log filtered by operation and target



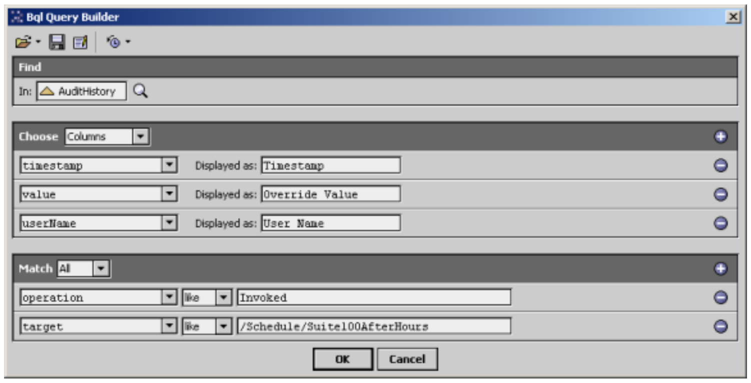
operation like Invoked
target like /Schedule/Suite100AfterHours

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

Figure 72 Creating query for the audit log filtered by operation, target and columns



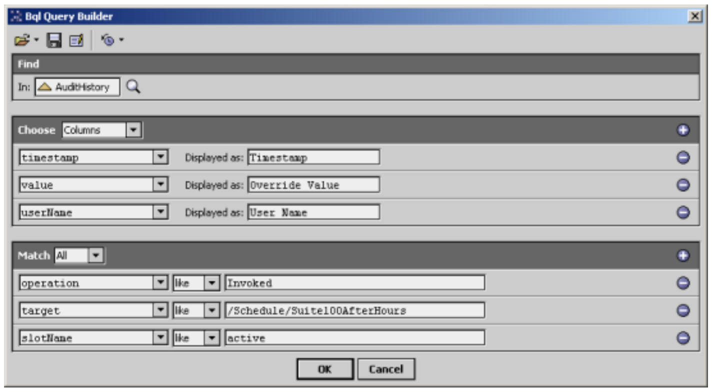
timestamp Displayed as: Timestamp
value Displayed as: Override Value
userName Displayed as: User Name
operation like Invoked
target like /Schedule/Suite100AfterHours

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

Figure 73 *Creating query for the audit log filtered by operation, target, columns and slot*



In: AuditHistory

Choose Columns

timestamp	Displayed as:	Timestamp
value	Displayed as:	Override Value
userName	Displayed as:	User Name

Match All

operation	like	Invoked
target	like	/Schedule/Suite100AfterHours
slotName	like	active

- 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 'Timestamp',value as
'Override Value',userName as 'User Name' where operation like 'Invoked' 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.

Weekly Electrical Demand Report

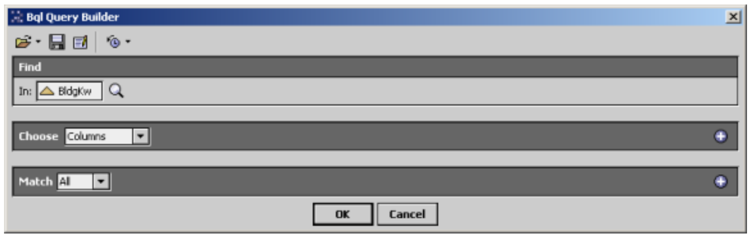
Bound tables and charts can be used in a report Px page to display historical information. An example might be to create a report which displays electrical demand readings for the previous week.

- 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 74 *Creating query for the BldgKw history*



In: BldgKw

Choose Columns

Match All

- 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.

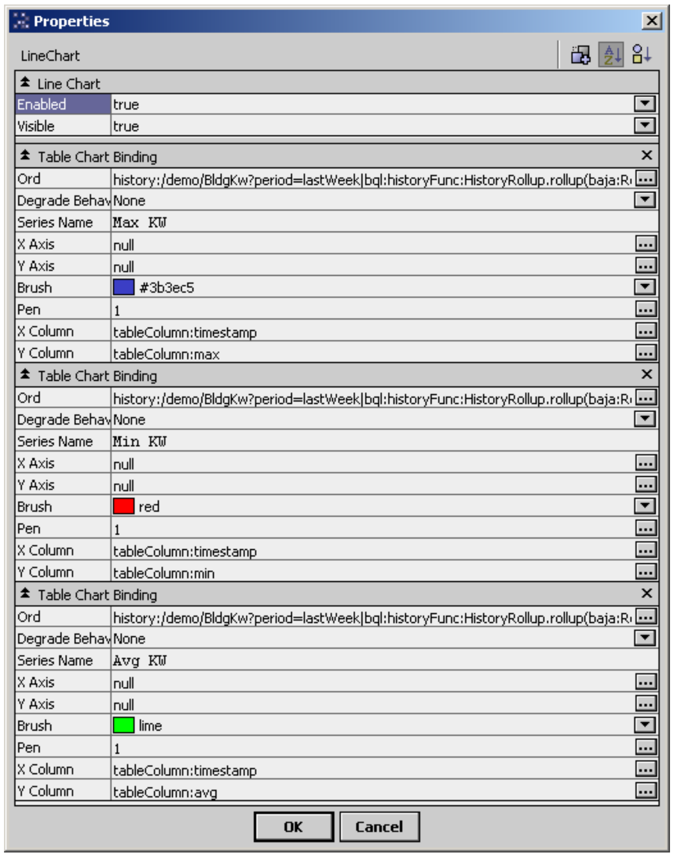
```
history:/demo/BldgKw?period=lastWeek|bql:historyFunc:HistoryRollup.rollup(baja:RelTime '86
```

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 75 *Creating table chart queries for the BldgKw history*



Line Chart	
Enabled	true
Visible	true
Table Chart Binding	
Ord	history:/demo/BldgKw?period=lastWeek|bql:historyFunc:HistoryRollup.rollup(baja:R
Degrade Behav	None
Series Name	Max KW
X Axis	null
Y Axis	null
Brush	🟦 #3b3ec5
Pen	1
X Column	tableColumn:timestamp
Y Column	tableColumn:max
Table Chart Binding	
Ord	history:/demo/BldgKw?period=lastWeek|bql:historyFunc:HistoryRollup.rollup(baja:R
Degrade Behav	None
Series Name	Min KW
X Axis	null
Y Axis	null
Brush	🟥 red
Pen	1
X Column	tableColumn:timestamp
Y Column	tableColumn:min
Table Chart Binding	
Ord	history:/demo/BldgKw?period=lastWeek|bql:historyFunc:HistoryRollup.rollup(baja:R
Degrade Behav	None
Series Name	Avg KW
X Axis	null
Y Axis	null
Brush	🟩 lime
Pen	1
X Column	tableColumn:timestamp
Y Column	tableColumn:avg

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.

Chapter 5 Scientific Notation Support

Topics covered in this chapter

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 NiagaraAX-3.3 displays values in scientific notation:

- E Notation Format
- Example Number Expressions

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
E signifies the exponent place, or base 10
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.

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 in NiagaraAX-3.3.

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.

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.

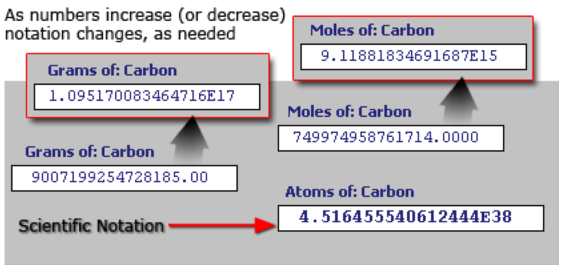


Figure 76 Displaying Large Numbers Using Scientific Notation

As numbers increase (or decrease) notation changes, as needed
Scientific Notation

Moles of: Carbon 9.11881834691687E15

Grams of: Carbon 1.0951700834647l6E17

Moles of: Carbon 749974958761714.0000

Grams of: Carbon 9007199254728185.00

Atoms of: Carbon 4.516455540612444E38

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.

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.

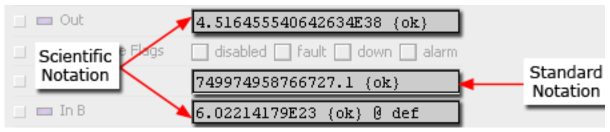


Figure 77 Scientific Notation in the Property Sheet View

```
4.516455540642634E38 {ok}
749974958766727.1 {ok}
6.02214179E23 {ok} @ def
```

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.

Chapter 6 BQL Expression component

Topics covered in this chapter

♦ Component features
♦ What the component is not
♦ Syntax
♦ Create a BQL Expression component
♦ Handling null
♦ Troubleshooting
♦ Frequently-asked questions

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 a AX-3.6 and later host.

Sections in this document describe the component and how to use it:

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.

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.

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

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

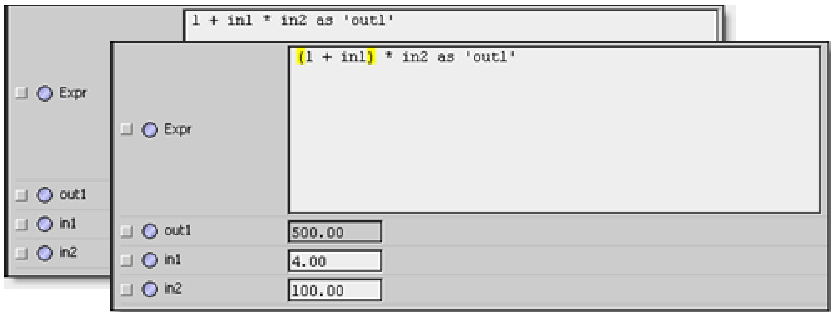
   comparisons
5. and, or

   logical operators
6. as

   result operator

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

Figure 78 *A change in precedence*



```
1 + in1 * in2 as 'out1'
```

```
(1 + in1) * in2 as 'out1'
```

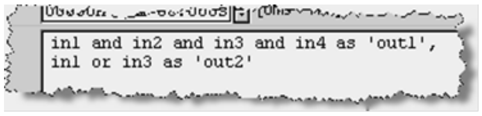
out1	500.00
in1	4.00
in2	100.00

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

Commas

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

Figure 79 *Example of commas*

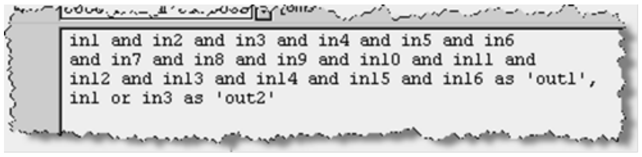


```
in1 and in2 and in3 and in4 as 'out1',
in1 or in3 as 'out2'
```

Long statements

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

Figure 80 *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.

Create a BQL Expression component

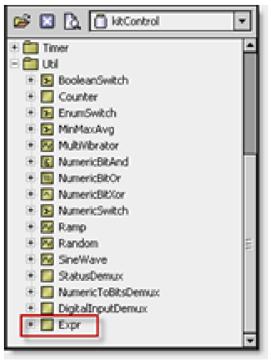
To create a BQL Expression component

Step 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.

Step 2 Drag an Expr from the kitControl palette onto your wire sheet and give it a name.

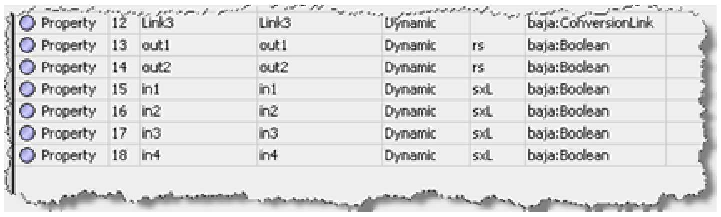
Figure 81 *kitControl components with BQL Expression (Expr) highlighted*



Step 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 82 *New slots (out1, out2, in1, in2, in3, in4)*



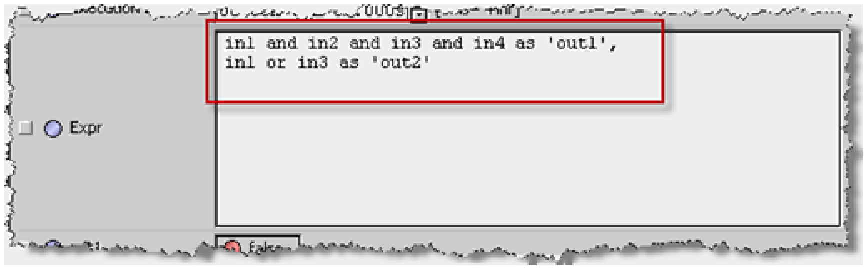
Property	12	Link3	Link3	Dynamic		baja:ConversionLink
Property	13	out1	out1	Dynamic	rs	baja:Boolean
Property	14	out2	out2	Dynamic	rs	baja:Boolean
Property	15	in1	in1	Dynamic	sxL	baja:Boolean
Property	16	in2	in2	Dynamic	sxL	baja:Boolean
Property	17	in3	in3	Dynamic	sxL	baja:Boolean
Property	18	in4	in4	Dynamic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

Step 4 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.

Step 5 Continuing on the property sheet, enter the BQL statements in the Expr field.

Figure 83 *Example expression statements*



```
in1 and in2 and in3 and in4 as 'out1',
in1 or in3 as 'out2'
```

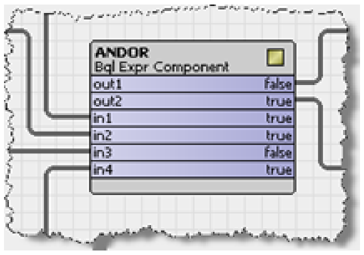
All other properties are assumed to be dynamic properties and can be added or removed on the Expr slot sheet.

Step 6 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 un-pinned slots. You can also use the right-click **Reorder** option to change the top-to-bottom positioning of slots.

Step 7 Using the wire sheet view, make links to the slots and test your logic.

Figure 84 *Example of an expected result*



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

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.

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: From Niagara 3.6 onwards, 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 the Engineering Notes document *NiagaraAX Conversion Links*.

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.

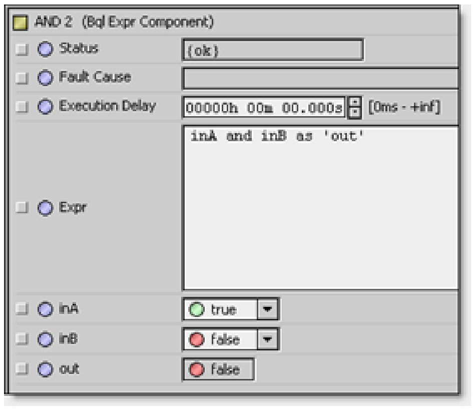
- Math folder

  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.

More examples

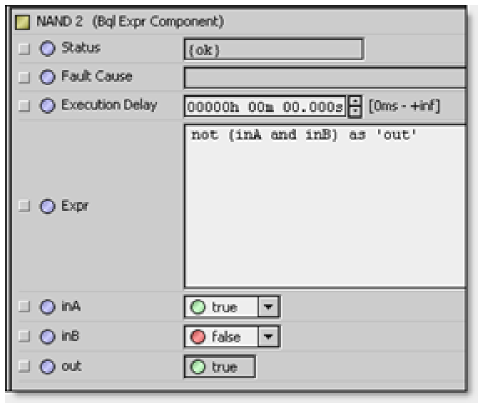
- Two inputs, logical AND



AND 2 (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	inA and inB as 'out'
inA	true
inB	false
out	false

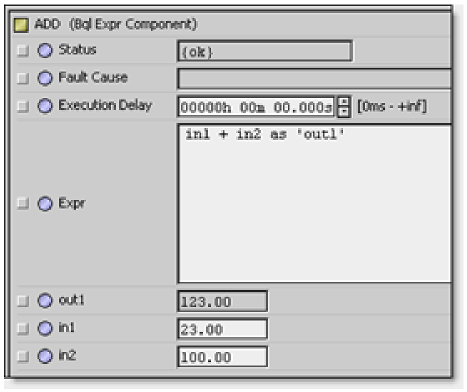
- Not two inputs



NAND 2 (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	not (inA and inB) as 'out'
inA	true
inB	false
out	true

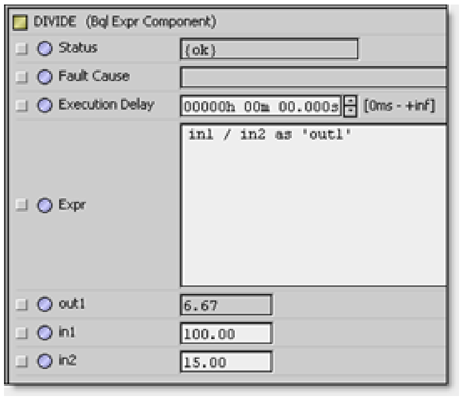
- Two-input addition



ADD (Bql Expr Component)

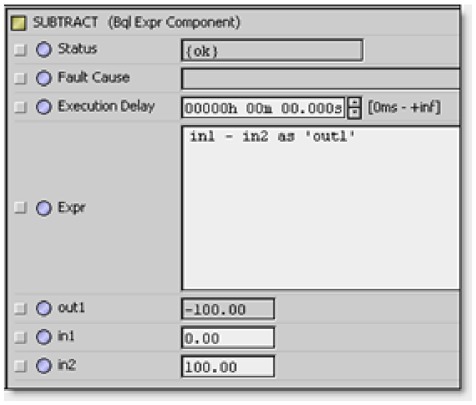
Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	in1 + in2 as 'out1'
out1	123.00
in1	23.00
in2	100.00

- Divider (two Double) properties



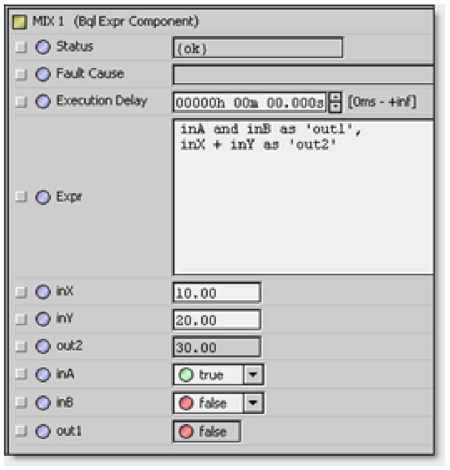
DIVIDE (Bql Expr Component)	
Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	in1 / in2 as 'out1'
out1	6.67
in1	100.00
in2	15.00

- Subtraction



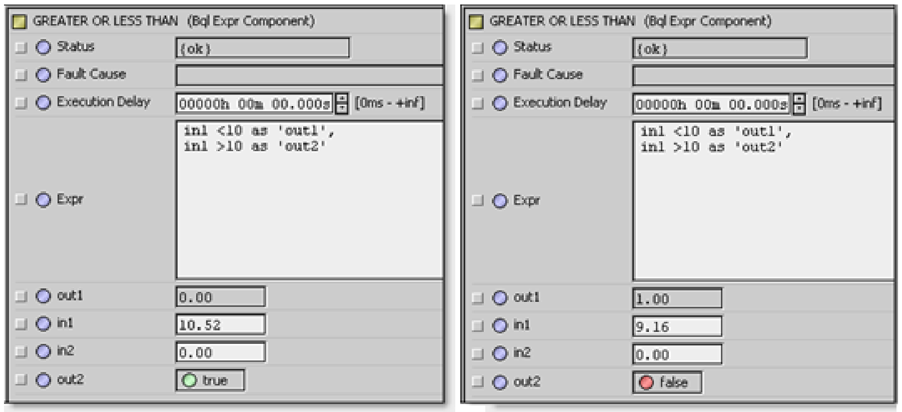
SUBTRACT (Bql Expr Component)	
Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	in1 - in2 as 'out1'
out1	-100.00
in1	0.00
in2	100.00

- Expression mixture



MIX 1 (Bql 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

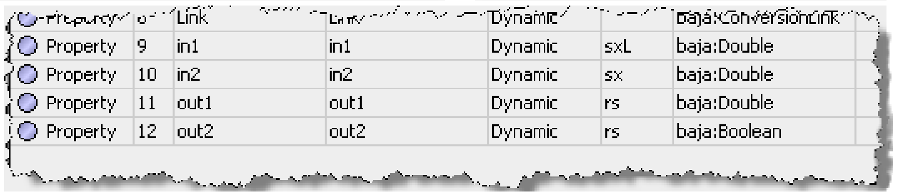


GREATER OR LESS THAN (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	inl <10 as 'outl', inl >10 as 'out2'
out1	0.00
in1	10.52
in2	0.00
out2	true

GREATER OR LESS THAN (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	inl <10 as 'outl', inl >10 as 'out2'
out1	1.00
in1	9.16
in2	0.00
out2	false

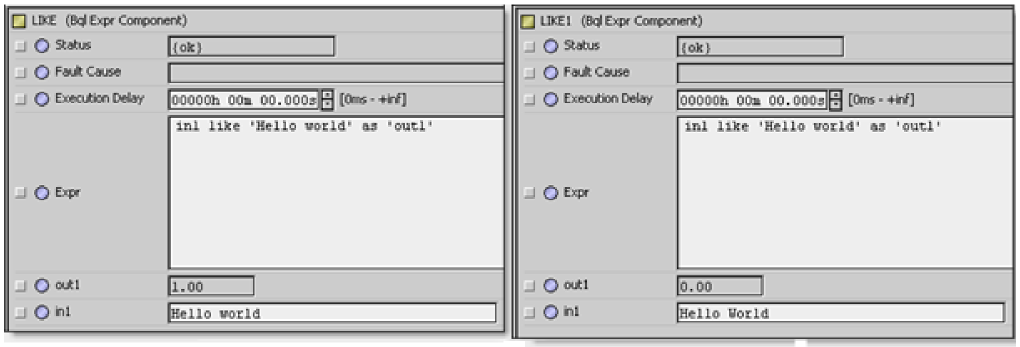


Property	8	Link	Link	Dynamic		baja:ConversionLink
Property	9	in1	in1	Dynamic	sxL	baja:Double
Property	10	in2	in2	Dynamic	sx	baja:Double
Property	11	out1	out1	Dynamic	rs	baja:Double
Property	12	out2	out2	Dynamic	rs	baja:Boolean

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

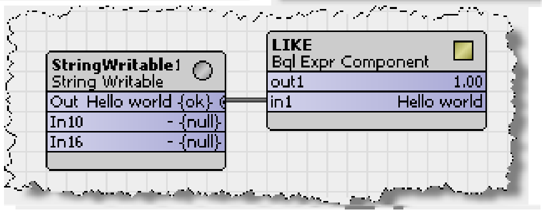


LIKE (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	inl like 'Hello world' as 'outl'
out1	1.00
in1	Hello world

LIKE1 (Bql Expr Component)

Status	{ok}
Fault Cause	
Execution Delay	00000h 00m 00.000s [0ms - +inf]
Expr	inl like 'Hello world' as 'outl'
out1	0.00
in1	Hello World



- 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'

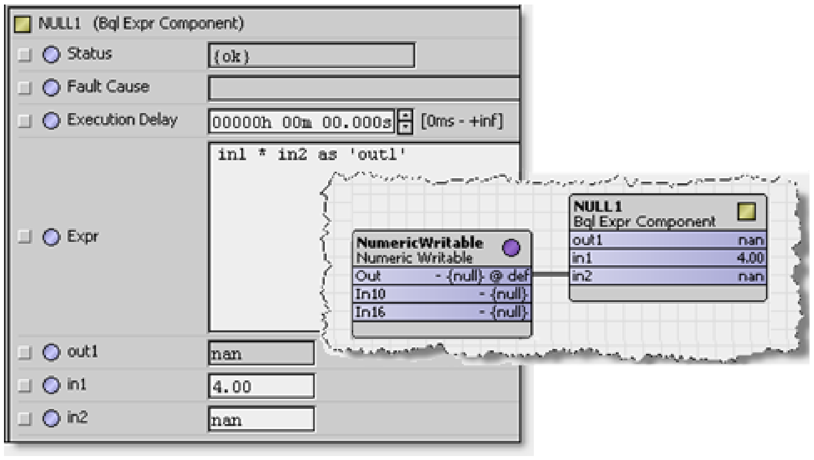
- Negate a Double input property (if inA is 5, out becomes -5)

  -inA as 'out'

Handling null

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

Figure 85 The result of a math expression with any null (or nan) input can be nan



```
Status            {ok}
Fault Cause
Execution Delay   00000h 00m 00.000s  [0ms - +inf]
Expr              in1 * in2 as 'out1'
out1              nan
in1               4.00
in2               nan
```

```
NumericWritable             NULL1
Out   - {null} @ def        out1   nan
In10  - {null}              in1    4.00
In16  - {null}              in2    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 the Engineering Notes document *NiagaraAX Conversion Links*.

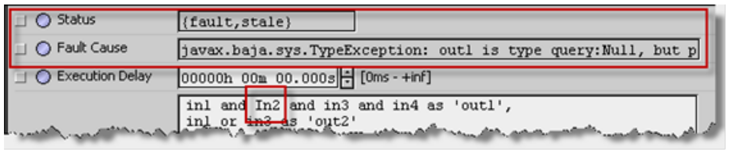
In general, if creating an Expr component for use in control logic that may have one or more “null” inputs, it is recommended that you test it to verify the desired behavior.

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”.

Troubleshooting

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

Figure 86 Example of a fault



```
Status            {fault,stale}
Fault Cause       javax.baja.sys.TypeException: out1 is type query:Null, but p
Execution Delay   00000h 00m 00.000s  [0ms - +inf]
                  in1 and In2 and in3 and in4 as 'out1',
                  in1 or in3 as 'out2'
```

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.

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 used 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.

Chapter 7   Conversion Links

Topics covered in this chapter

♦ Conversion link usage
♦ Supported conversion link types
♦ Converter components
♦ Converter properties
♦ Conversion link "From" notes

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.

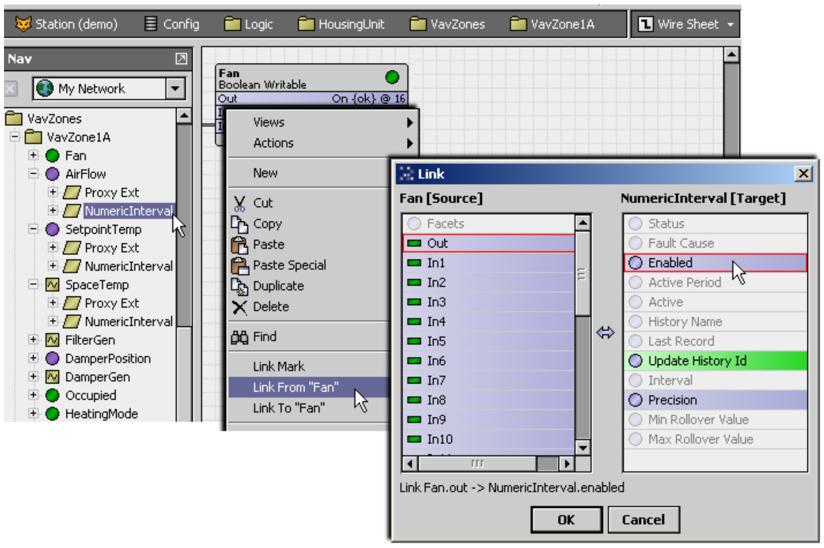
NOTE: Conversion links require AX-3.6 or higher on “both ends” when working with engineering control logic. That is, the host (JACE) running the station must be at AX-3.6 or higher, and Workbench must be at AX-3.6 or higher. Otherwise, conversion links are unavailable.

Conversion link usage

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 87   Example link from status type to simple type (statusBoolean to Boolean)

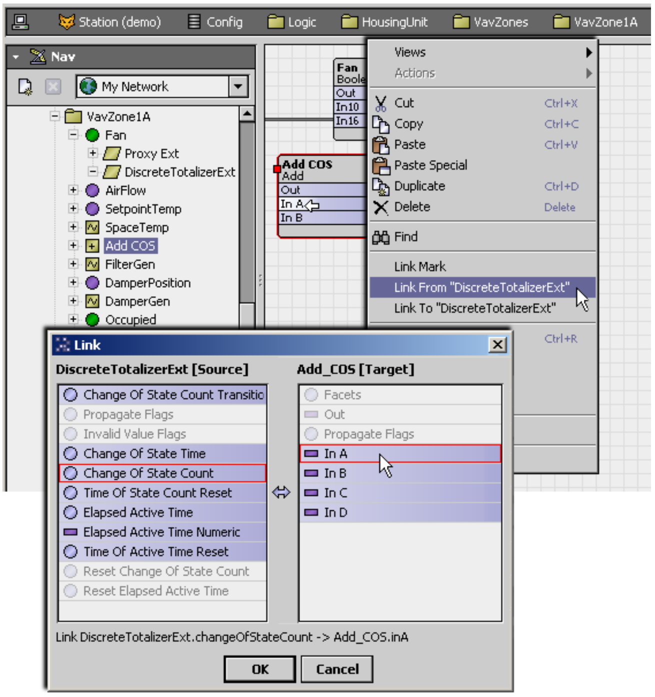


Link Fan.out -> NumericInterval.enabled

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 88   Example link from simple type to status type (integer to statusNumeric)



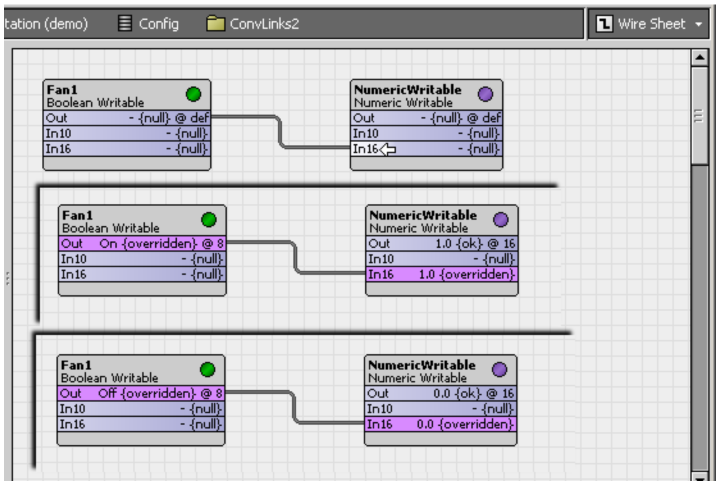
Link DiscreteTotalizerExt.changeOfStateCount -> Add_COS.inA

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.

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 89   Example link from statusBoolean to statusNumeric

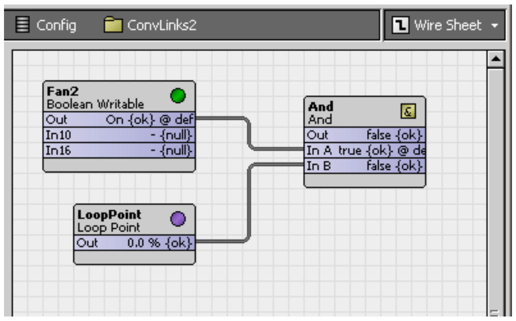


In the example above, the statusBoolean "out" of a BooleanWritable is linked to the statusNumeric "In16" slot of another component. With no other configuration, this transforms the boolean "active" value to a numeric "1", a boolean "inactive" value to a numeric "0", and passes through a "null" value.

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 90 *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" on page 4 for a matrix of allowed links by data types.
- "Converter components" on page 5 for details on the converter that is automatically selected.
- "Converter properties" on page 6 for information on possible converter properties.
- "Conversion link "From" notes" on page 7 for details on linking from some specific data types.

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	ord	time	absTime	relTime
To	string	—	Y*	Y*	Y*	Y*	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	—	Y	Y	Y
	float	Y	Y	Y	—	Y	Y	Y	Y	Y*	Y	Y	Y	—	Y	Y	Y
	long	Y	Y	Y	Y	—	Y	Y	Y	Y*	Y	Y	Y	—	Y	Y	Y
	integer	Y	Y	Y	Y	Y	—	Y	Y	Y*	Y	Y	Y	—	Y	Y	Y

		From															
	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	—	—	Y	Y
	statusEnum	—	Y	Y	Y	Y	Y	Y	Y	Y	Y	—	—	—	—	—	—
	statusString	Y	Y	Y*	Y*	Y*	Y*	Y	Y	Y	Y*	Y	—	Y	—	Y*	Y
	ord	Y	—	—	—	—	—	—	—	—	—	—	Y	—	—	—	—
	time	—	—	Y	Y	Y	Y	—	—	—	—	—	—	—	—	Y	—
	absTime	Y	—	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.

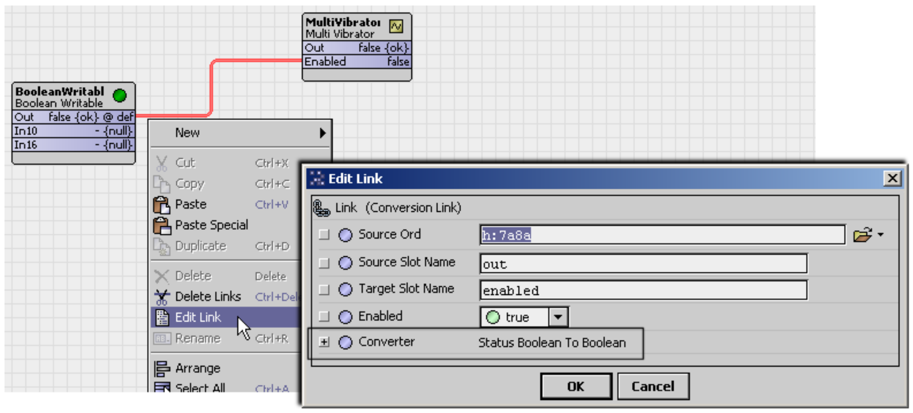
NOTE: Y* indicates that the Converter component of the link has one or more properties.

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

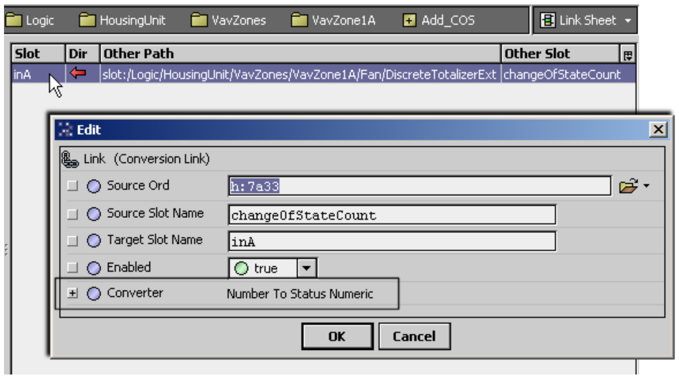


Link (Conversion Link)
Source Ord: h:7a8a
Source Slot Name: out
Target Slot Name: enabled
Enabled: true
Converter: Status Boolean To Boolean

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 92 Go to link sheet of component and double-click conversion link for "Edit" dialog



Slot	Dir	Other Path	Other Slot
inA	⇐	slot:/Logic/HousingUnit/VavZones/VavZone1A/Fan/DiscreteTotalizerExt	changeOfStateCount

Link (Conversion Link)

Source Ord	h:7a33
Source Slot Name	changeOfStateCount
Target Slot Name	inA
Enabled	true
Converter	Number To Status Numeric

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 in NiagaraAX. 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".

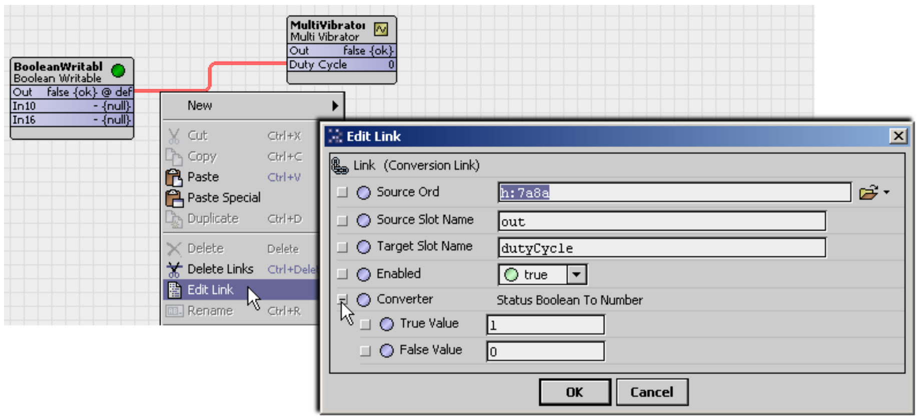
Another possible exception may be with some conversion links, where a child Converter component has one or more properties (such links are indicated with a "Y*" in the Table 1 matrix of supported links). See "Converter properties" on page 6.

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 93 Example Converter with properties (Status Boolean To Number)



Link (Conversion Link)

Source Ord	h:7a8a
Source Slot Name	out
Target Slot Name	dutyCycle
Enabled	true
Converter	Status Boolean To Number
True Value	1
False Value	0

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".

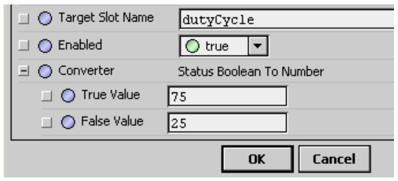
Expanding the Converter component in the **Edit Link** dialog reveals two properties:

- 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 94 *Example Converter properties edited to non-default values*

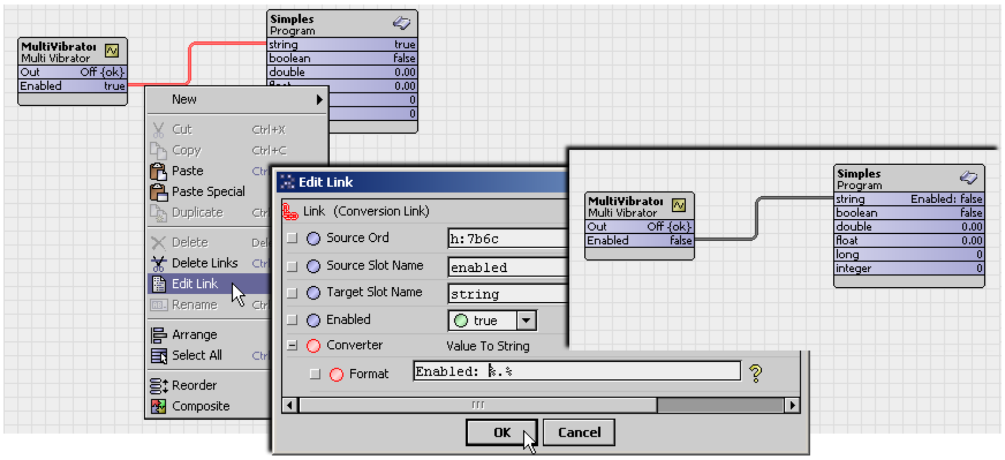


Target Slot Name	dutyCycle
Enabled	true
Converter	Status Boolean To Number
True Value	75
False Value	25

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 95 *Example Format property of "Boolean To String" Converter edited from default "%.%"*



Edit Link

Link (Conversion Link)

Source Ord	h:7b6c
Source Slot Name	enabled
Target Slot Name	string
Enabled	true
Converter	Value To String
Format	Enabled: %.%

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

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

- 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 sting 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 sting 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" on page 7.

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" on page 7.

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" on page 7.

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" on page 7.

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" on page 7.

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" on page 7.

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 sting 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 sting 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 Niagara 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" on page 7.

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" on page 7.

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" on page 7.

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`.

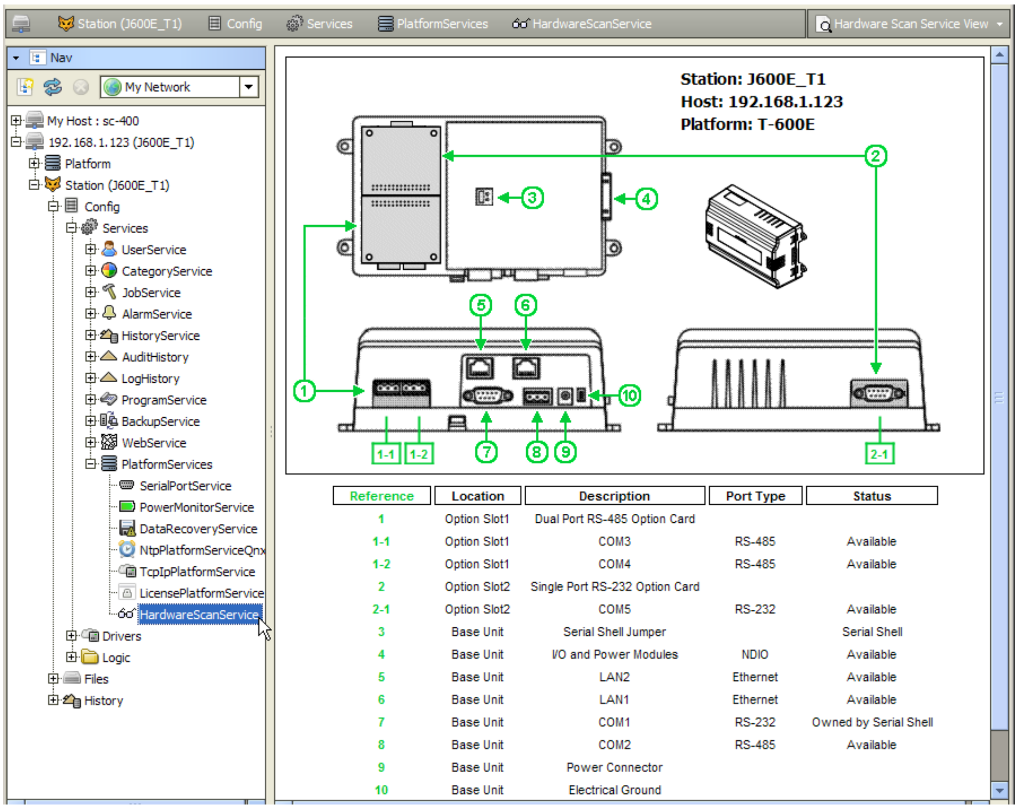
Links to other "Conversion link "From" notes" on page 7.

Chapter 8 JACE Hardware Scan Service

Topics covered in this chapter

Starting in AX-3.7, NiagaraAX support was added for a "Hardware Scan Service" to be added under a JACE station's PlatformServices. This optional service has a default view that shows a diagram of the hosting JACE controller, identifying the location of communication ports and other features.

Figure 96 Example Hardware Scan Service View for a JACE-6E



Reference	Location	Description	Port Type	Status
1	Option Slot1	Dual Port RS-485 Option Card		
1-1	Option Slot1	COM3	RS-485	Available
1-2	Option Slot1	COM4	RS-485	Available
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
5	Base Unit	LAN2	Ethernet	Available
6	Base Unit	LAN1	Ethernet	Available
7	Base Unit	COM1	RS-232	Owned by Serial Shell
8	Base Unit	COM2	RS-485	Available
9	Base Unit	Power Connector		
10	Base Unit	Electrical Ground		

Included are callouts to a table that explain the description (such as COM2), port type, and status.

Also included are what types of JACE option cards are installed in option card slots (if applicable), plus other information on various controller features, such as the current "Serial Shell" jumper position.

Hardware scan benefits

NOTE: The HardwareScanService is unlicensed, requiring only a JACE controller running AX-3.7 or later.

Prior to this service, the hardware configuration of a 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 a JACE-2, JACE-202 Express (M2M JACE), or a Security JACE.

Although a station’s platform SerialPortService lists each serial port on a 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.

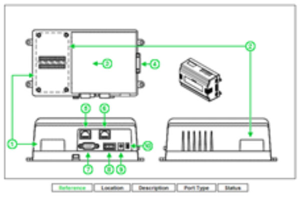
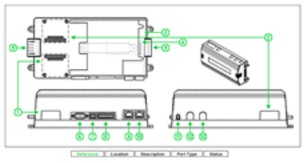
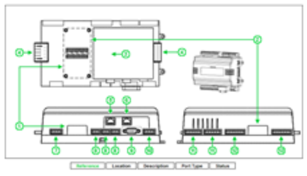
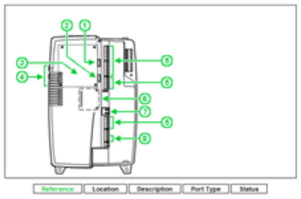
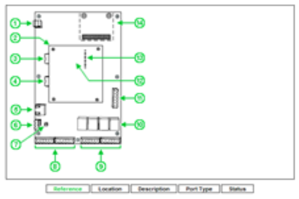
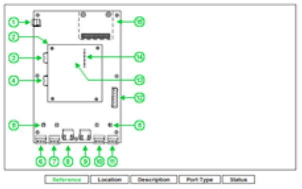
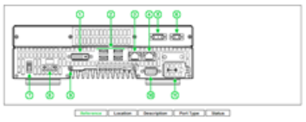
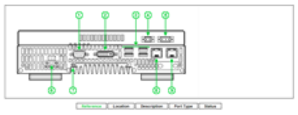
The Hardware Scan Service clarifies all of this by providing a complete hardware profile of the hosting JACE controller to its station and Workbench. Included is the controller’s “Product Model” type (combination of its base board and NPM module, if applicable) with a representative image, along with details about any installed JACE 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 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.

Currently supported platforms

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

JACE-2, JACE-3E, JACE-6, JACE-6E	JACE-7 (JACE-700)	Security JACE (201 and 601)
		
JACE-x02 Express (M2M JACE)	JACE-603	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” on page 3.

Adding the HardwareScanService

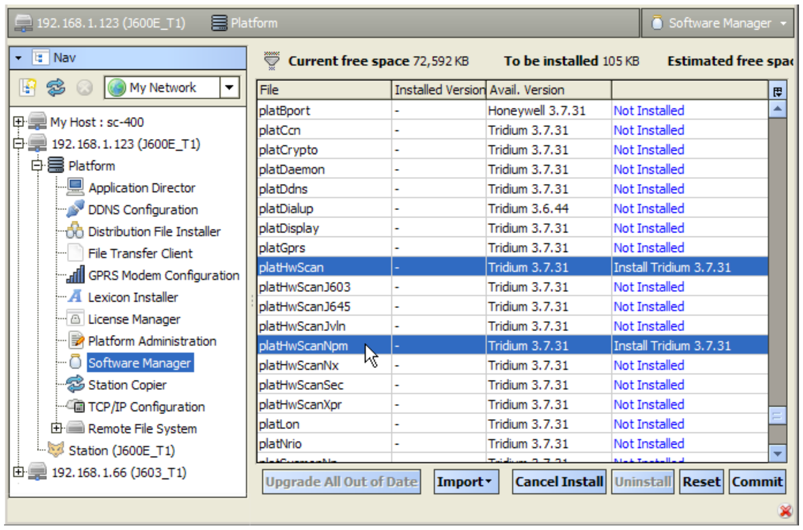
You can add the Hardware Scan Service in any JACE controller running AX-3.7 or later. To add the service, simply install two software modules, using the **Software Manager** in a platform connection:

- **platHwScan** (always necessary), plus:
- **platHwScanType**

where Type varies by the specific JACE model series (see below). If you select only the `platHwScanType` module, the required `platHwScan` module becomes automatically selected.

NOTE: Installing these modules results in a reboot of the JACE controller.

Figure 97 Choosing the two platHwScan modules needed for a JACE-6, JACE-3E, or JACE-6E series


192.168.1.123 (J600E_T1) Platform Software Manager

Nav

My Network

- My Host : sc-400
- 192.168.1.123 (J600E_T1)
  - Platform
    - Application Director
    - DDNS Configuration
    - Distribution File Installer
    - File Transfer Client
    - GPRS Modem Configuration
    - Lexicon Installer
    - License Manager
    - Platform Administration
    - Software Manager
    - Station Copier
    - TCP/IP Configuration
    - Remote File System
  - Station (J600E_T1)
- 192.168.1.66 (J603_T1)

Current free space 72,592 KB **To be installed** 105 KB **Estimated free spac**

File	Installed Version	Avail. Version	
platBport	-	Honeywell 3.7.31	Not Installed
platCcn	-	Tridium 3.7.31	Not Installed
platCrypto	-	Tridium 3.7.31	Not Installed
platDaemon	-	Tridium 3.7.31	Not Installed
platDdns	-	Tridium 3.7.31	Not Installed
platDialup	-	Tridium 3.6.44	Not Installed
platDisplay	-	Tridium 3.7.31	Not Installed
platGprs	-	Tridium 3.7.31	Not Installed
platHwScan	-	Tridium 3.7.31	Install Tridium 3.7.31
platHwScanJ603	-	Tridium 3.7.31	Not Installed
platHwScanJ645	-	Tridium 3.7.31	Not Installed
platHwScanJvln	-	Tridium 3.7.31	Not Installed
platHwScanNpm	-	Tridium 3.7.31	Install Tridium 3.7.31
platHwScanNx	-	Tridium 3.7.31	Not Installed
platHwScanSec	-	Tridium 3.7.31	Not Installed
platHwScanXpr	-	Tridium 3.7.31	Not Installed
platLon	-	Tridium 3.7.31	Not Installed
platNrio	-	Tridium 3.7.31	Not Installed

The image above shows the two modules selected that are necessary for a JACE-2, JACE-3E, JACE-6, or JACE-6E series controller. By controller series, platHwScanType is as follows:

platHwScanType module needed, by controller model series

Controller Series	platHwScanType module
JACE-6, JACE-3E, JACE-6E	`platHwScanNpm`
JACE-7 (700)	`platHwScanJvln`
JACE-603 (retrofit board)	`platHwScanJ603`
JACE-645 (retrofit board)	`platHwScanJ645`
JACE-x02 Express (202/602-XPR or M2M)	`platHwScanXpr`
JACE-NXS/JACE-NXT	`platHwScanNx`
SEC-J-601	`platHwScanSec`

NOTE: If you install the 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.

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 to a table below with item details.

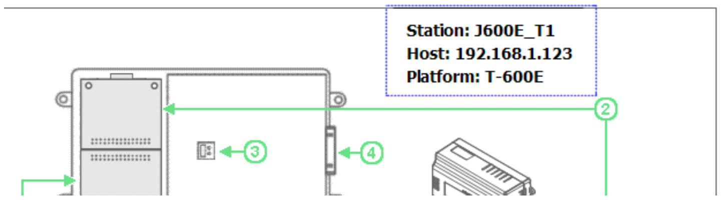
The following sections describe features of this view:

- Text in image notes
- Option card notes
- Callout to table notes
- Px usage of Hardware Scan Service View

Text in image notes

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

Figure 98 *Text on three lines near top of image portion in Hardware Scan Service View*


Station: J600E_T1
Host: 192.168.1.123
Platform: T-600E

- Station: Station_Name

  The Niagara station name of the station running on the JACE.
- Host: IP_Address

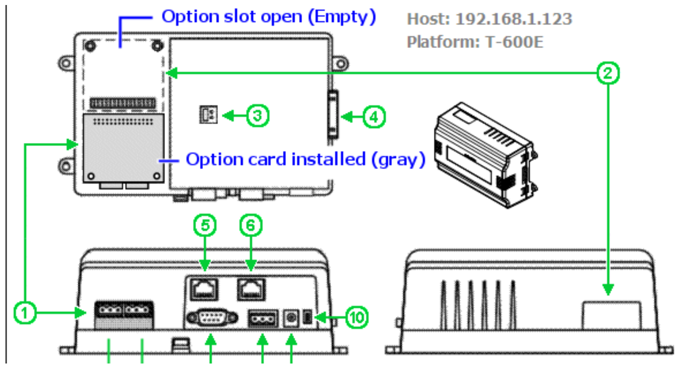
  The IP address of the JACE used in this Fox (station) connection from Workbench.
- Platform: Product_Model

  "Product Model" descriptor, which may vary according to the vendor of the JACE controller.

Option card notes

Most JACEs support one or two JACE-6-type 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 99 *Installed option cards are shaded gray*

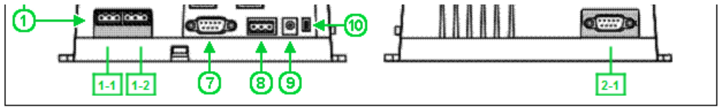


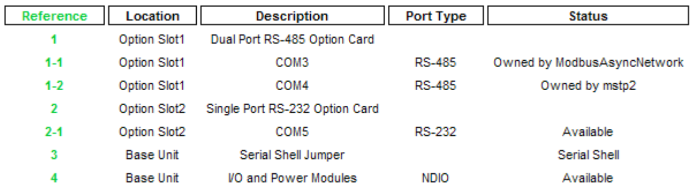
A numerical callout to the table below describes any installed option card. In the case of a JACE-700, which has a MiniPCI slot in addition to two option card slots, any installed WiFi (MiniPCI) option is also indicated by gray shading.

Callout to table notes

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

Figure 100 *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, 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

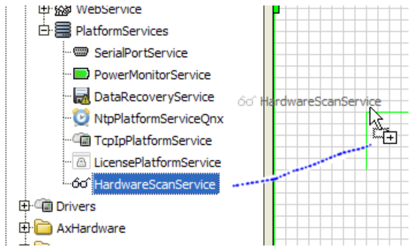
BacnetNetwork), or even `Serial Shell` for a COM1 port—if the serial shell jumper is installed. If a serial port not currently used in the station, status is `Available`.

- 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`.

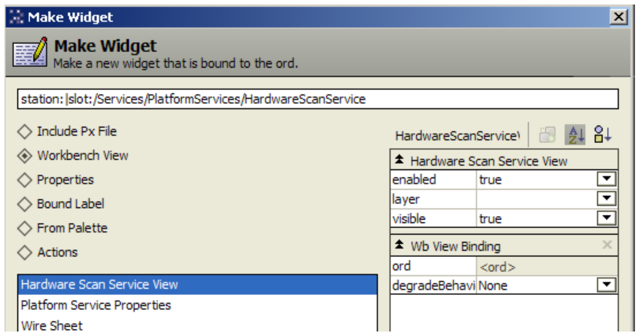
Px usage of Hardware Scan Service View

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 101 Dragging the HardwareScanService onto a Px view



Make 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. Depending on the controller series, the overall "footprint" (pixel dimensions) varies. The approximate width and height dimensions for each view, including the lower "hardware reference table" area plus side scroll bar, are provided in Table 2.

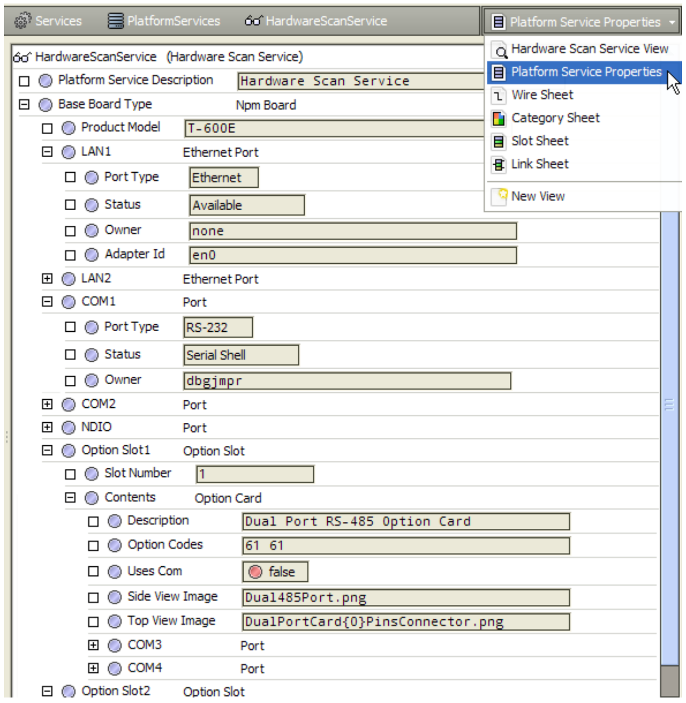
Pixel dimensions of different Hardware Scan Service Views, by controller series

Controller Series	Width x Height (pixels)
JACE-6 (600), JACE-3E, or JACE-6E	660 x 720
JACE-7 (700)	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

HardwareScanService properties

The HardwareScanService has a number of properties, available by selecting its property sheet from the view selector (Figure 8). Values from many of these properties are reflected in the graphical default view.

Figure 102 Property sheet for example HardwareScanService



```
HardwareScanService  (Hardware Scan Service)
  Platform Service Description   Hardware Scan Service
  Base Board Type                Npm Board
    Product Model                T-600E
    LAN1                         Ethernet Port
      Port Type                  Ethernet
      Status                     Available
      Owner                      none
      Adapter Id                 en0
    LAN2                         Ethernet Port
    COM1                         Port
      Port Type                  RS-232
      Status                     Serial Shell
      Owner                      dbgjmpr
    COM2                         Port
    NDIO                         Port
    Option Slot1                 Option Slot
      Slot Number                1
      Contents                   Option Card
        Description              Dual Port RS-485 Option Card
        Option Codes             61 61
        Uses Com                 false
        Side View Image          Dual485Port.png
        Top View Image           DualPortCard{0}PinsConnector.png
        COM3                     Port
        COM4                     Port
    Option Slot2                 Option Slot
```

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.

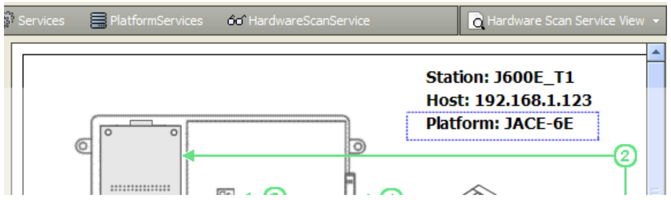
As with most other station platform services, there is an available “Poll” action on the HardwareScanService. However, usage should rarely be needed, due to the static nature of hardware configuration.

Lexicon customizing of HardwareScanService

Customizing text descriptors and values that appear in the service’s Hardware Scan Service View is possible 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:

```
NPM6E=JACE-6E
```

The “Platform=” value near the top of the controller image will be “JACE-6E” (instead of “T-600E”) when viewing the service in a JACE-6E, as shown below.

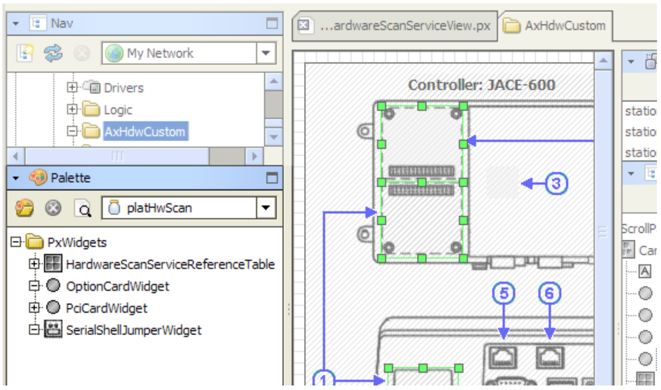


NOTE: Before such changes can become effective, you need to save and install the lexicon changes in the target JACE, and also reboot that controller.

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 103 Palette of platHwScan module has four Px widgets



As shown, 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).

- PciCardWidget

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

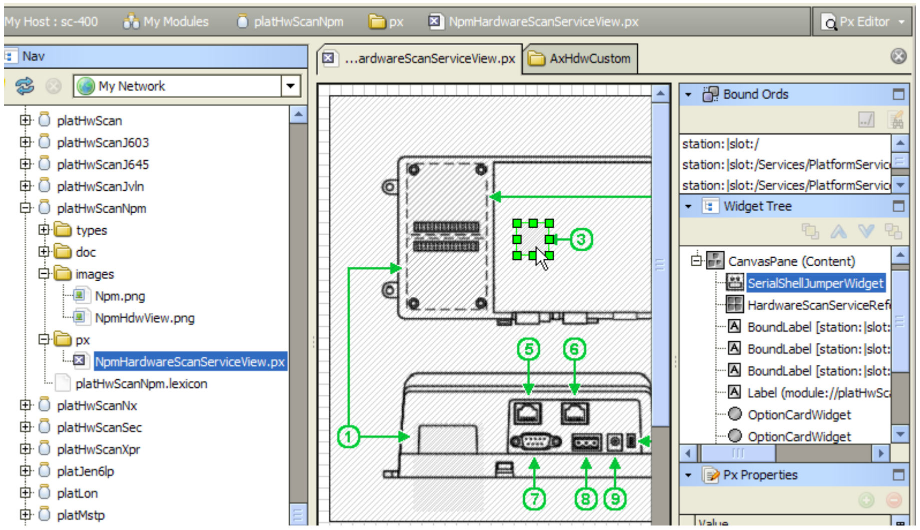
- SerialShellJumperWidget

  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.

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 Workbench PC. Do this by expanding a `platHwScanType` module to look at the contents of its px folder in the PxEditor (edit mode).

Figure 104 *Looking at the Px view in the platHwScanNpm module*

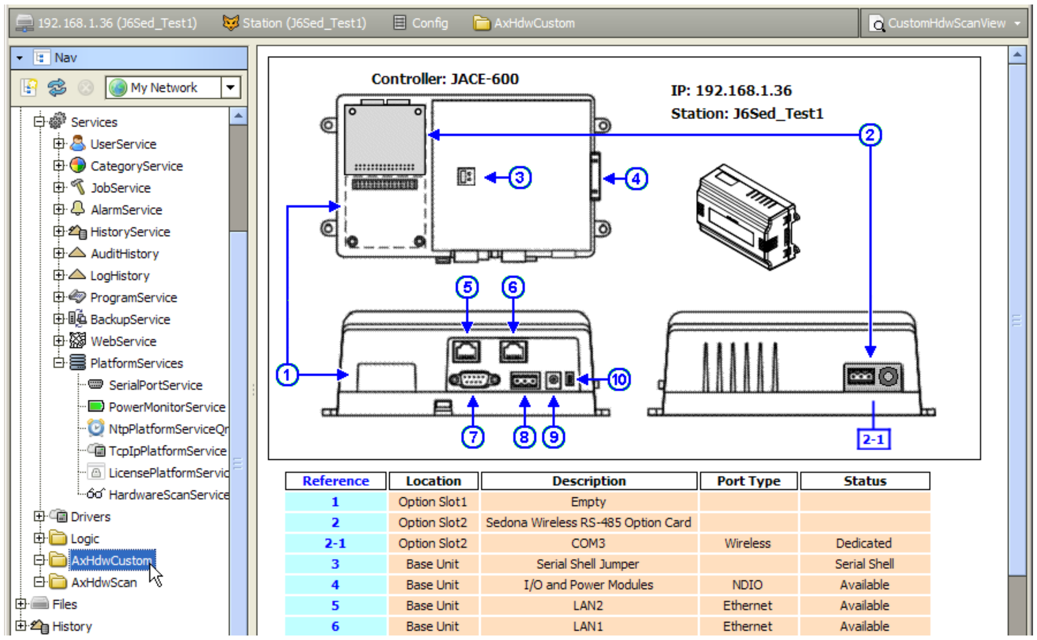


Look at widget properties to see how layout parameters and other items are set.

Example customized Px view for Hardware Scan Service

An example customized Px view for a JACE-6, JACE-3E, or JACE-6E series is shown.

Figure 105 *Example customized Px view for Hardware Scan Service*



Reference	Location	Description	Port Type	Status
1	Option Slot1	Empty		
2	Option Slot2	Sedona Wireless RS-485 Option Card		
2-1	Option Slot2	COM3	Wireless	Dedicated
3	Base Unit	Serial Shell Jumper		Serial Shell
4	Base Unit	I/O and Power Modules	NDIO	Available
5	Base Unit	LAN2	Ethernet	Available
6	Base Unit	LAN1	Ethernet	Available

In this example, the controller image (NpmHdwView.png) was copied from the `platHwScanNpm` module and then edited in a graphics program to “color replace” the callout arrows and numbers from the default green to blue. The edited png file was 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.

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 “arrowless” 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 more tailored view of the Hardware Scan Service information.

Chapter 9 Formats (BFormat)

Topics covered in this chapter

- ♦ BFormat default values
- ♦ Example scenarios
- ♦ BFormat errors

Baja data type or baja-Format (BFormat), is a class used to format objects into strings using a standardized formatting pattern language. Bformats are very important for making templates (reusable portions of the data model Tree that require minimal configuration), when creating Px views, and to enable localization (foreign language support).

A format string is normal text with an embedded scrip denoted by the % percent character (use %% to insert a real %). A script is one or more calls chained together using the dot/period (.) operator. The system resolves formats (executes the instructions contained in the embedded script) starting with the object that declares the format. Embedded calls then dynamically resolve to objects, and the system converts the final object into a string.

In Niagara, many properties that allow text entry use the embedded BFormat scripts. For example, a formula can refer to multiple points by using a Value Ord that contains BFormat script:
`slot:/Drivers/NiagaraNetwork/%parent.name%/points/%name%`

The name of the folder containing the point is substituted for the first script, and the name of the point is substituted for the second script.

The system resolves calls in this order:

1. Special calls

   The following are special calls:

   - time() returns the current time as a Niagara BAbsTime object
   - user() returns the current user's name
   - lexicon(module:key) get the specified lexicon text
   - decodeFromString(module:type:escapedEncodedValue)
   - substring(to) on a string
   - substring(-fromEnd) on a string
   - substring(from,to) on a string
2. Java method get<call>(Context)
3. Java method get<call>()
4. Java method <call>()
5. Java method get("<call>")

BFormat default values

As copied from palettes or originated from manager views, some components already have default values in certain BFormat-type properties (while others may default as empty).

This table lists examples of the components with default values.

Table 2 Default values for a few properties using BFormat

Component	Property	Default Value	Notes
Alarm extension for points, e.g. OutOfRangeExt, etc.	sourceName	%parent.displayName%	Suitable as is in many cases, such as where all parent points are uniquely named.
History extension for points, e. g. NumericInterval, etc.	historyName	%parent.name%	
Any network component's AlarmSourceInfo slot.	sourceName	%parent.displayName%	Often both properties work with these default values.
Any device-level component's AlarmSourceInfo slot.	sourceName	%parent.parent.displayName% % parent.displayName%	
EmailRecipient	subject	Niagara Alarm From %alarmData. sourceName%	The system provides much additional alarm data in the slot that contains the body of the email.

For the property value you can use multiples of scripted variables along with static text, as demonstrated by the defaults for a device's **`AlarmSourceInfo`** (`sourceName`) property in the table. A static space character separates the `%parent.parent.displayName%` from `%parent.displayName%`. The subject property of the EmailRecipient contains static text `Niagara Alarm From` ahead of the variable.

With BFormat variables (scripting), you can sometimes save time by enabling the replication of applications, where desired results happen with minimal custom edits to property values on your part. While reducing engineering time, enabling the replication of applications can still yield consistent output results. Or, you may have specific text formatting needs.

Example scenarios

Example scenarios demonstrate how to work with different non-default values for the BFormat-type properties.

Alarm extensions, history extensions, Px Widgets, and WeatherService are good examples to illustrate how to use BFormat.

BFormat example: naming points

This example uses the BFormat parent.parent method to name points in alarm extensions.

VAV scenario

You have a driver network of VAVs for 60 zones using 60 identical devices, each with identically-named proxy points, but under a uniquely-named device component. For simplicity, assume that the devices are named: VAV1, VAV2, …to VAV60.

Several proxy points in each zone require an alarm extension. You could: manually rename the points, or type unique values in the BFormat type properties under each alarm extension, as well as related properties (in some cases) under its **`offNormalAlgorithm`** slot. This would ensure that alarm records (viewed in the alarm console) would show unique **Source** values for any alarm, without having to decipher by station ord (for example) which RoomTemp point was in alarm, for example to isolate by zone.

Manually renaming the points in each zone

This would involve entering unique values in the BFormat type properties under each alarm extension.

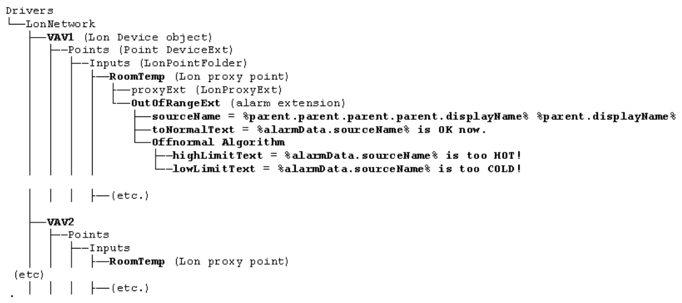
In some cases, you may also have to modify related properties under the device's **`offNormalAlgorithm`** slot. These changes would ensure that alarm records viewed in the alarm console provide unique source values (names) for each alarm. For example, without a unique source and only the station ord, it can be difficult to isolate the zone and RoomTemp point that is in alarm.

It could take you at least several hours to manually configure 60 zones.

Replicating properties

This method will save you time. You begin by replacing the default values for several BFormat-type properties of one VAV's alarm extension such that generated alarms contain more useful source data, then replicate the VAV application to the remaining 59 VAVs.

Figure 106 *Example config structure and alarm extension property values using edited BFormat-type data*



```
Drivers
└─LonNetwork
    ├──VAV1 (Lon Device object)
    │   ├--Points (Point DeviceExt)
    │   │   ├--Inputs (LonPointFolder)
    │   │   │   ├──RoomTemp (Lon proxy point)
    │   │   │   │   ├──proxyExt (LonProxyExt)
    │   │   │   │   └──OutOfRangeExt (alarm extension)
    │   │   │   │       ├──sourceName = %parent.parent.parent.parent.displayName% %parent.displayName%
    │   │   │   │       ├──toNormalText = %alarmData.sourceName% is OK now.
    │   │   │   │       └──Offnormal Algorithm
    │   │   │   │           ├--highLimitText = %alarmData.sourceName% is too HOT!
    │   │   │   │           └--lowLimitText = %alarmData.sourceName% is too COLD!

    │   │   │   ├──(etc.)
    │
    ├──VAV2
    │   ├--Points
    │   │   ├--Inputs
    │   │   │   ├──RoomTemp (Lon proxy point)
  (etc)
 .  │   │   │   ├──(etc.)
```

The above shows part of the station's config structure, including the non-default values entered for BFormat-type properties under one alarm extension in one of the identically named points. The `sourceName` of the alarm extension has been changed to use two variables:

- %parent.parent.parent.parent.displayName%
- %parent.displayName%

These variables are separated by a space.

Given the tree structure in use, the alarm record shows four (parent) levels up for the first part of the source, that is the device (for example. VAV1), then the proxy point name appears as the second part of the source, for example RoomTemp. So, in the alarm console the alarm source displays as "VAV1 RoomTemp".

Here is the tricky part: All the alarm text properties are relative to the *alarm record* component generated by an alarm, and *not* to the alarm extension responsible for generating the alarm. Alarm text properties in the alarm extension **`Offnormal Algorithm`** include:

- **`toNormalText`**
- **`toOffNormalText`**

If the extension is an **`OutOfRangeExt`**, alarm text properties include:

- **`highLimitText`**
- **`lowLimitText`**

  These **`OutOfRangeExt`** properties override any entry in the **`toOffNormalText`** property of the alarm extension parent.

The location of these text properties in the alarm record means that you cannot use the parent.displayName scheme for the alarm text properties, at least not if you expect to get any useful results.

But because each alarm extension's sourceName is now unique (using the technique above), you can reference it within alarm text type properties, along with any desired static text. Except here, the sourceName is an alarmData field, from the alarm record.

In the example , when RoomTemp in VAV1 triggers a high limit alarm, the alarm data message text is: `VAV1 RoomTemp is too HOT!`, and when it returns to normal the alarm data message text is: `VAV1 RoomTemp is OK now`.

You could further modify the **`OffnormalAlgorithm`** high and low limit text properties to include the numerical (alarm) limit, using another alarmData field. For example, if **`highLimitText`** is set to a BFormat value of `%alarmData.source% is above %alarmData.highLimit% degrees!`, and the extension's **`highLimit`** is set to 74.5, upon a high limit alarm the message text generated is `VAV1 RoomTemp is above 74.5 degrees!`. This technique may be useful if routing the alarm using a minimum amount of alarm data text, say including only the timestamp and the message text.

NOTE: To see what `alarmData` fields are available for use in this manner, go to a station's alarm console and view the complete details (**Alarm Record** popup) for any one alarm.

BFormat example: naming histories

This example uses a technique other than the parent.parent method to name histories. This method may be called the folder-level independent method, as explained in this topic.

History extension scenario

Consider a VAV network with replicated device applications. In each VAV zone, you need histories of several proxy points, all identically named. For example, in each zone you wish to have a numeric interval history on RoomTemp and another one on Damper. However, you must either rename the parent point(s) or rename the history extensions' `historyName` to something unique, as duplicate history Ids are forbidden.

Or before replicating this VAV application, you could edit the `historyName` in all history extensions, similar to editing the sourceName when naming points.

Figure 107   Example config structure and history extension property values using edited BFormat-type data

```
Drivers
└─LonNetwork
  ├──VAV1 (Lon Device object)
  │  ├─-Points (Point DeviceExt)
  │  │  ├─-Inputs (LonPointFolder)
  │  │  │  ├──RoomTemp (Lon proxy point)
  │  │  │  │  ├──proxyExt (LonProxyExt)
  │  │  │  │  └──numericInterval (history extension)
  │  │  │  │     └──historyName = %parent.proxyExt.device.displayName%_%parent.displayName%
  │  │  │  ├──(etc.)
  │  │  │
  │  │  ├─-Damper (Lon proxy point)
  │  │  │  ├──proxyExt (LonProxyExt)
  │  │  │  │  └──numericInterval (history extension)
  │  │  │  │     └──historyName = %parent.proxyExt.device.displayName%_%parent.displayName%
  │  │  │  ├──(etc.)
  │  │  │
  │  │  ├─-Outputs (LonPointFolder)
  │  │  │  ├──(etc.)
```

The above shows part of the station's config structure, including the non-default value entered in place of-`historyName` for two history extensions.

The `historyName` for each extension has been changed to use two variables:

- %parent.proxyExt.device.displayName%
- %parent.displayName%

These names are separated by an underscore. This syntax uses a special getDevice() method in the first variable, where the proxy point's parent device name is resolved, regardless of its folder depth under the Points extension. (In this example, proxy point RoomTemp is in an `Inputs` point subfolder, while the Damper proxy point is in the root of the `Points` extension).

Because the points are located in different folders, the parent.parent method would work for RoomTemp, but not for Damper. The method folder-level independent method, however, is more fault tolerant as a result of moving a proxy point, especially to change its hierarchy.

Given the tree structure of this network, the resulting histories appear as `VAV1_RoomTemp`, `VAV1_Damper`, and if replicated, `VAV2_RoomTemp`, `VAV2_Damper`, and so on.

Here is how this works: "%parent.proxyExt.device.displayName%", the parent steps up one level to the Lon proxy point (say, RoomTemp). The **`proxyExt`** is the slot name that walks back down the tree to a different child component, in this case to the **`LonProxyExt`**. The device calls the getDevice() method, and the `displayName` calls the getDisplayName() method.

This example assumes that no other components in the station also have a VAV" component with a RoomTemp child, which also requires a history extension. Also, the example uses an underscore instead of a space even though spaces in object names are permitted (history being one type of object), they are escaped in the database using a "%20" string. This can be confusing in certain scenarios.

BFormat Px Widget examples

BFormat can be used with Px widgets as shown in this example topic.

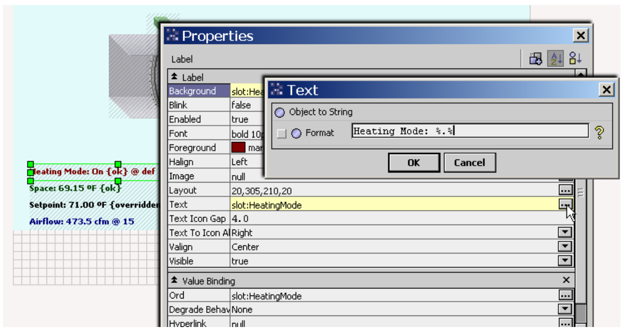
Px widget scenarios

When engineering Px widget properties, especially for BoundLabel types with a binding to a component, there is an important property that uses the BFormat type:

The BFormat: `BoundLabel:Text (ObjectToString):` determines the content of the displayed text.

By default, when you drag a component onto a Px page (for example a BooleanWritable point), the **`Make Widget`** wizard associates a BoundLabel's `Text` with (and makes a binding to) the component *ord*. The default text value is: %.%

Figure 108 Example of adding static text in Text property of a BoundLabel



Heating Mode: On {ok} @ def
Space: 69.15 ºF {ok}
Setpoint: 71.00 ºF {overridden
Airflow: 473.5 cfm @ 15

Format: `Heating Mode: %.%`

Label	
Background	slot:Hea
Blink	false
Enabled	true
Font	bold 10
Foreground	mar
Halign	Left
Image	null
Layout	20,305,210,20
Text	slot:HeatingMode
Text Icon Gap	4.0
Text To Icon Al	Right
Valign	Center
Visible	true
Value Binding	
Ord	slot:HeatingMode
Degrade Behav	None
Hyperlink	null

In the example, a simple edit adds "Heating Mode:" in front of this default `Text` value, as shown below.

Default BoundLabel Text results

For any point (or any component with an **`Out`** property), default text from the binding is identical to the out value displayed in the component's property sheet.

The default text that is identical to the **`Out`** value includes facets, as well as the following information:

- If bound to a *writable* point, the `Text` contains three pieces of data, namely:

```
<value> <status> @priorityLevel>
```

  where:

  - `<value>` is
  - `<status>` is {ok}, etc.
  - `@jpriorityLevel>` is

  For example: `On {ok} @16` (a BooleanWritable) or `20% {ok} @12` (a NumericWritable)

- If bound to a read-only point, the default `Text` provides two pieces of data in the text, that is:

```
<value> <status>
```

  For example:

  `Clean {ok}` (a BooleanPoint) or `72.3 °F {ok}` (a NumericPoint)

- If bound to a component that is not a point (there is no **`Out`** property), you must bind to a particular *slot* of that component, in order to display text other than its component type.

For example, if you drag a **`DegreeDays`** component to a Px page, the system displays the default text: `Degree Days`. However, if you change the binding's ord to *<objectName>*/clgDegDays, the system calculates and displays the cooling degree-days value (and status), such as: `5.0 {ok}`

Editing BoundLabel Text for points

You can edit the `Text` property in any BoundLabel widget to include additional static text, and/or modify (or limit) the real-time data in the text.

The following table provides a few example BFormatting variables and results for writable points.

Text (BFormat) value	Description	Example 1	Example 2 (script and result)
%out.value%	Value only (with facets).	On	`AHU is %out.value%` AHU is On.
%out.status%	Status including priority level, if writable point.	{ok} @ 16	`Status of AHU is %out.status%.` Status of AHU is {ok} @ 16.
%activeLevel%	Number only (1-16, def) for priority level, writable points only.	16	`AHU is %out.value% at level %activeLevel%.` AHU is On at level 16.
%status.flagsToString%	String value(s) for status flags set, without braces. If non: ok.	ok	`AHU status is %status.flagsToString%.` AHU status is ok.

The **Example 1** column illustrates the resulting text if the **`Out`** script is `On {ok} @ 16`.

The **Example 2** column shows the script and the resulting display when static text is added to the script.

Advanced BoundLabel Text editing

You can use the Object to String BFormat script with BoundLabel.

Object-to-String scripting is quite flexible when working with BoundLabel widgets. You are limited only by your understanding of Baja (see online Bajadoc in the Help system). For the non-developer, these few simple rules may help:

- The BoundLabel widget must actually be bound to an object (using an ord). In other words, you cannot simply drag a BoundLabel from the **kitPx** palette to the Px page, edit the**`Text`** property, and get results. If needed, the Text value may be totally unrelated to the bound object. For example, you can bind to any object and enter a system-type call, as shown here:

```
%time().toDateString% to produce text like "01-Nov-08"
```

- Relative to the bound object, you can use the parent technique to "walk up" the component tree of the text for a slot (or name), for example %parent.parent.name% gives the name of the parent two levels up. For example, a BoundLabel bound to a **`DiscreteTotalizerExt`** under a **`BooleanPoint`**, where you wish to display the (parent) point's name and the number of times it has changed state since its last reset, could be achieved using this **`Text`** entry:

```
%parent.displayName% had %changeOfStateCount% COS since last reset.
```

  The resulting text might be: `ChWPump2 had 14 COS since last reset.`.

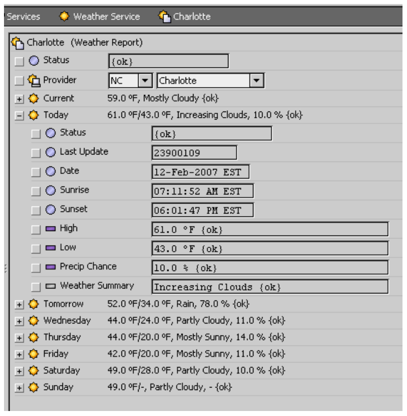
- In addition, relative to the bound object, you can also "walk down" the tree in a parallel path, using the `slot name` (versus `name` or `displayName`).An example of this "walk down" method (via slot name) is in the history extension (`historyName`) example, along with the parent technique.

BFormat: WeatherService example

This example uses the WeatherService to show how to use BFormat.

The WeatherService can provide many pieces of information including current conditions and forecasts. The following image shows the property sheet for a weather report with one locale under the weather service.

Figure 109 *Example Weather Report property sheet*



Services | Weather Service | Charlotte

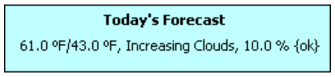
Charlotte (Weather Report)

Status	{ok}
Provider	NC / Charlotte
Current	59.0 ºF, Mostly Cloudy {ok}
Today	61.0 ºF/43.0 ºF, Increasing Clouds, 10.0 % {ok}
Status	{ok}
Last Update	23900109
Date	12-Feb-2007 EST
Sunrise	07:11:52 AM EST
Sunset	06:01:47 PM EST
High	61.0 °F {ok}
Low	43.0 °F {ok}
Precip Chance	10.0 % {ok}
Weather Summary	Increasing Clouds {ok}
Tomorrow	52.0 ºF/34.0 ºF, Rain, 78.0 % {ok}
Wednesday	44.0 ºF/24.0 ºF, Partly Cloudy, 11.0 % {ok}
Thursday	44.0 ºF/20.0 ºF, Mostly Sunny, 14.0 % {ok}
Friday	42.0 ºF/20.0 ºF, Mostly Sunny, 11.0 % {ok}
Saturday	49.0 ºF/28.0 ºF, Partly Cloudy, 10.0 % {ok}
Sunday	49.0 ºF/-, Partly Cloudy, - {ok}

To display this information on a graphic you create a bound label that references the WeatherService and the applicable property. For example, to display the forecast for today, the referenced ord would be `station:|slot:/Services/WeatherService/Charlotte/day0`.

Instead of entering this value, you could expand the WeatherService in the Nav tree to find this slot, then drag it to the Px page, where the **`Make Widget`** wizard automatically resolves to this ord. If the format `Text` is left at the default of `%.%`, the text values that appear as shown in the second line in the following image.

Figure 110 *Default BoundLabel Text to Weather Report's Today property*



Today's Forecast

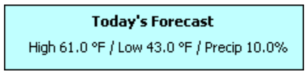
61.0 ºF/43.0 ºF, Increasing Clouds, 10.0 % {ok}

To control whether they appear and provide additional text and formatting, you can individually reference each of the **`Today`** properties in the format `Text`.

For example, setting `Text` to the following:

```
High %High.value% °F / Low %Low.value% °F / Precip %PrecipChance.value%%%
```
results in the graphic displaying text as shown in the second line of the following image.

Figure 111 *Example modified BoundLabel Text to Weather Report's Today property*



Today's Forecast

High 61.0 ºF / Low 43.0 ºF / Precip 10.0%

NOTE: The system uses the percentage symbol (%) To delimit scripts in format `Text` fields. To display this symbol as text, enter two of them (%%).

BFormat errors

This topic covers a few example errors and considerations for dealing with errors.

Not all attempts at customizing Format-type property values may be successful. If a syntax error causes the script to fail, an `ERROR` or `err:<item>`, where `<item>` is the script value appears in the produced text.

For example:

- If you forget a % on BoundLabel `Text` entry, say: `Fan is %out.value`, the system displays: `ERROR Fan is %out.value`.
- Or, a script call to a misnamed slot might fail with a displayed error similar to: `ChwPump2 had %err:control:DiscreteTotalizerExt:changeOfStates% COS since last reset`.

Make sure you test all modifications to BFormat-type properties, to make sure you get the intended results.

Chapter 10 bacnetUtil Component Usage

Topics covered in this chapter

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 explains some differences and documents the new bacnetUtil module.

These topics are covered:

- Broadcast message management, page 119 explains how BBMD works and documents the two BBMD changes implemented in Niagara 4: you are required to manually enter the desired BBMDs into the BDT (there is no automatic configuration); and you may have more than one BBMD per subnet.
- BACnet Ethernet on Windows using WinPcap, page 120 has been added to explain how to download and install WinPcap, which is required for Windows platforms.
- A new module, BACnetUtil, has been added for troubleshooting connections. BACnet troubleshooting, page 121 begins the information about BACnetUtil.

Broadcast message management

Use of BBMDs (BACnet Broadcast Management Devices) makes it possible for a BACnet network to span multiple IP subnets. When using BBMDs, each IP subnet with BACnet/IP devices requires at least one BBMD. Multiple BBMDs are allowed.

A BBMD may be a device operating solely as a BBMD or, more typically, a device that includes BBMD functions in addition to other application or controller functions. A station running a BACnet network and configured for BBMD is a multi-function device.

Why and what a BBMD does

A BBMD supports delivery of BACnet broadcast messages, such as Who-Is and Who-Has. As a rule, the standard IP routers used to connect separate IP subnets block globally broadcast messages. (Standard IP routers do not block directed messages between devices on different subnets, such as common ReadProperty requests.)

Working with other peer BBMDs, each BBMD acts as a broadcast manager for its local subnet. Stored in each BBMD is a table that links the BBMD to its peer BBMDs. This BDT (Broadcast Distribution Table) contains the IP address and distribution mask of other BBMDs in the system, itself included. Each subnet may have multiple BBMDs. You are responsible for determining which BBMDs to put in any given BDT. Consequently, the BDT in each BBMD may differ from the BDT in every other BBMD.

In a small configuration with only a few (or perhaps just one) BACnet/IP devices, the use of a local BBMD is not necessary for broadcast message support. As an alternative, the devices in the subnet may register as foreign devices with the BBMD of a remote subnet. The registration process adds the local device to a second table in the remote BBMD, the FDT (Foreign Device Table). Using this table, the remote BBMD takes the responsibility of delivering global BACnet broadcast messages to the device(s) on the local subnet.

The term “foreign device” implies no stigma or limitation of service. It simply makes possible the management of messages when a BBMD is not practical.

Registering as a foreign device is sometimes used for BACnet/IP devices that are temporarily connected. Consequently, the registration process requires the assignment of a registration lifetime in seconds (**`Time To Live`**) for each foreign device. The device must be re-registered within the **`Time To Live`** or the BBMD purges the device from its FDT. This prevents unnecessary broadcast delivery attempts to part-time participants.

The FDT reflects the current list of registered foreign devices along with each device's **`Time To Live`** value and calculated purge time. If a station is configured as a BBMD, you can see all these entries in the BBMD's FDT.

You modify a BDT and FDT using the **Bacnet Comm Property Sheet**. When adding a device, you must manually update the BDT. There is no automatic configuration.

When all devices on a local subnet, including the BBMD, receive a globally broadcast message, they reply as needed to the broadcast device without BBMD involvement. Then, the local BBMD forwards the broadcast message to the other subnets using one of two methods (as defined for each remote subnet):

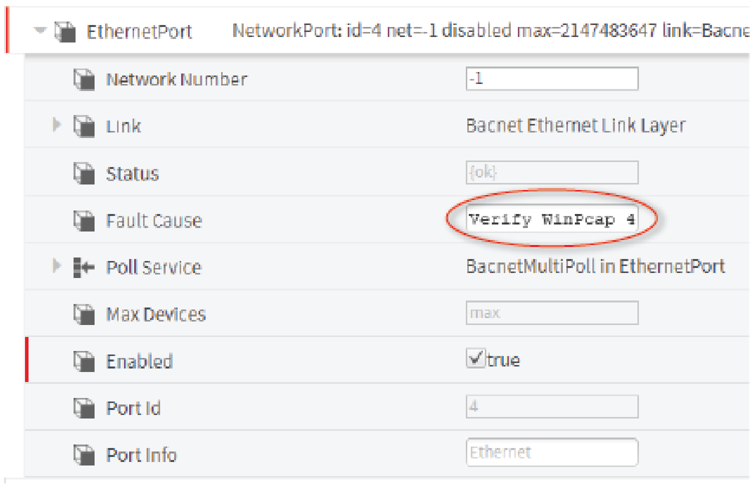
- Two-hopThe BBMD sends the message to its peer BBMD on the remote subnet and the remote BBMD broadcasts the message on its local subnet. This is the normal case in a system with multiple subnets.
- One-hopThe BBMD directly broadcasts the message to another subnet IP router and the target router broadcasts the message to all devices on its local subnet. This method is unusual as IP routers rarely pass directed broadcasts on to the local subnet.

Replies to BACnet broadcast messages may or may not require BBMD involvement. Occasionally, replies are directed messages back to the particular BACnet/IP device that generated the message. However, replies to a Who-Is broadcast are often broadcast I-Am messages. This is the case with Niagara

BACnet Ethernet on Windows using WinPcap

Starting a version of Niagara that contains a BACnet/Ethernet adapter without having WinPcap installed causes the **`NetworkPort`** to go into fault. The system reports the fault cause as: "Verify WinPcap 4.1.x is installed."

Figure 112 *Fault message when WinPcap is not installed*



EthernetPort	NetworkPort: id=4 net=-1 disabled max=2147483647 link=Bacne
Network Number	-1
Link	Bacnet Ethernet Link Layer
Status	{ok}
Fault Cause	Verify WinPcap 4
Poll Service	BacnetMultiPoll in EthernetPort
Max Devices	max
Enabled	✓ true
Port Id	4
Port Info	Ethernet

WARNING: Sending and receiving raw Ethernet frames on a Windows platform requires a Kernel Mode Driver. Anyone with malicious intent who connects to a system that has the raw Ethernet Kernel Mode Driver installed can send and receive raw Ethernet frames to assert full control over your BACnet/Ethernet network. This is the case with all raw Ethernet drivers. If you are not willing to accept this increased attack risk, upgrade to BACnet/IP.

WinPcap must be manually downloaded and installed to continue using BACnet/Ethernet on supported Niagara 4 Windows platforms.

Step 1 Confirm that the `EthernetPort` is enabled.

Step 2 Stop the station.

Step 3 Download WinPcap from https://www.winpcap.org/install/.

Step 4 Double-click the downloaded installer to begin the installation process and follow the **Setup Wizard**.

Step 5 When prompted, leave "`Automatically start the WinPcap driver at boot time`" selected.

Step 6 When the wizard finishes installing, restart the station.

When you start the station with an enabled BACnet `EthernetPort`, WinPcap is used to send and receive packets.

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

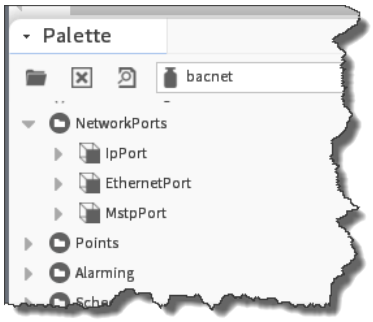
bacnetUtil provides tools to monitor the health of the BACnet ports in a station, and diagnose port problems. Three `bacnetUtil` tools count tokens and messages.

Prerequisites: Your BACnet network has been set up and all devices are configured and communicating.

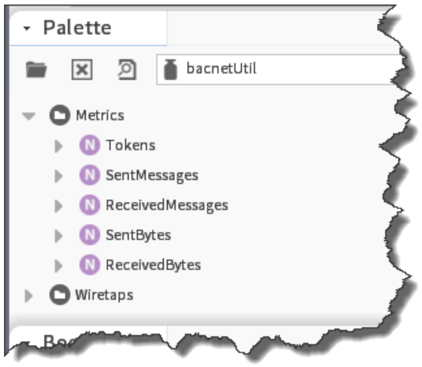
Some networks do not allow the Wireshark utility to be installed on their control networks. Investigating certain BACnet communication problems without proper network diagnostic tools can be difficult. The **bacnetUtil** module is intended to provide diagnostic resources for these networks.

Step 1 To prepare the Nav tree, expand **Config→Drivers→BacnetNetwork→Bacnet Comm→Network**.

Step 2 To set up an MS/TP port, open the **bacnet** (Niagara BACnet Driver) palette and expand **NetworkPorts**.



Step 3 Drag the **MstpPort** container to the **Network** container in the Nav tree.



Step 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.

Step 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

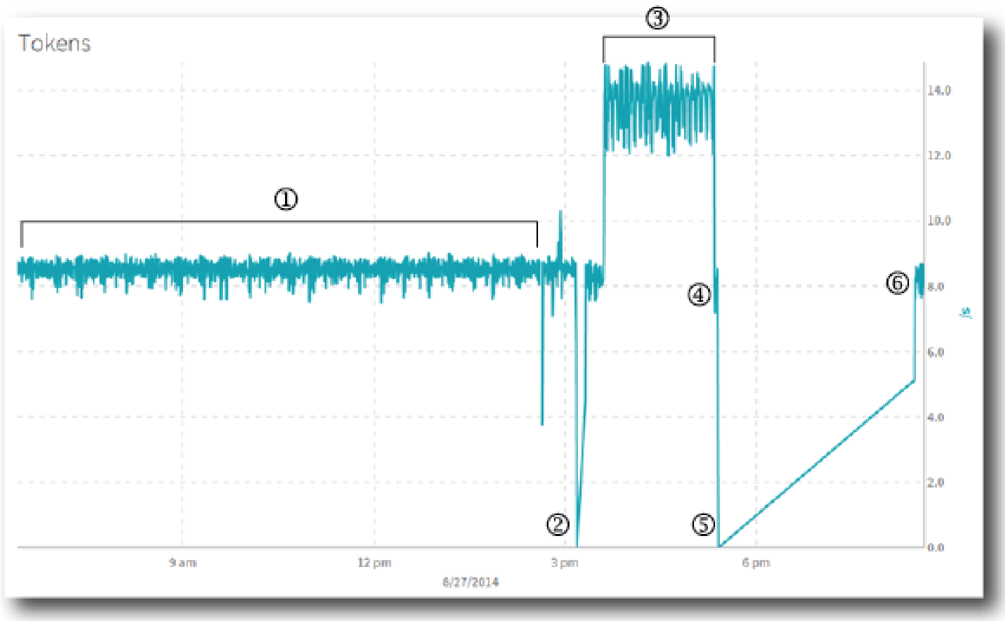
The Tokens component aids in diagnosing certain configuration problems. While the problems may require a site visit to resolve, the information gathered should aid in diagnosis and repair.

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.

Figure 113 *Example of a Tokens graphic*



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).

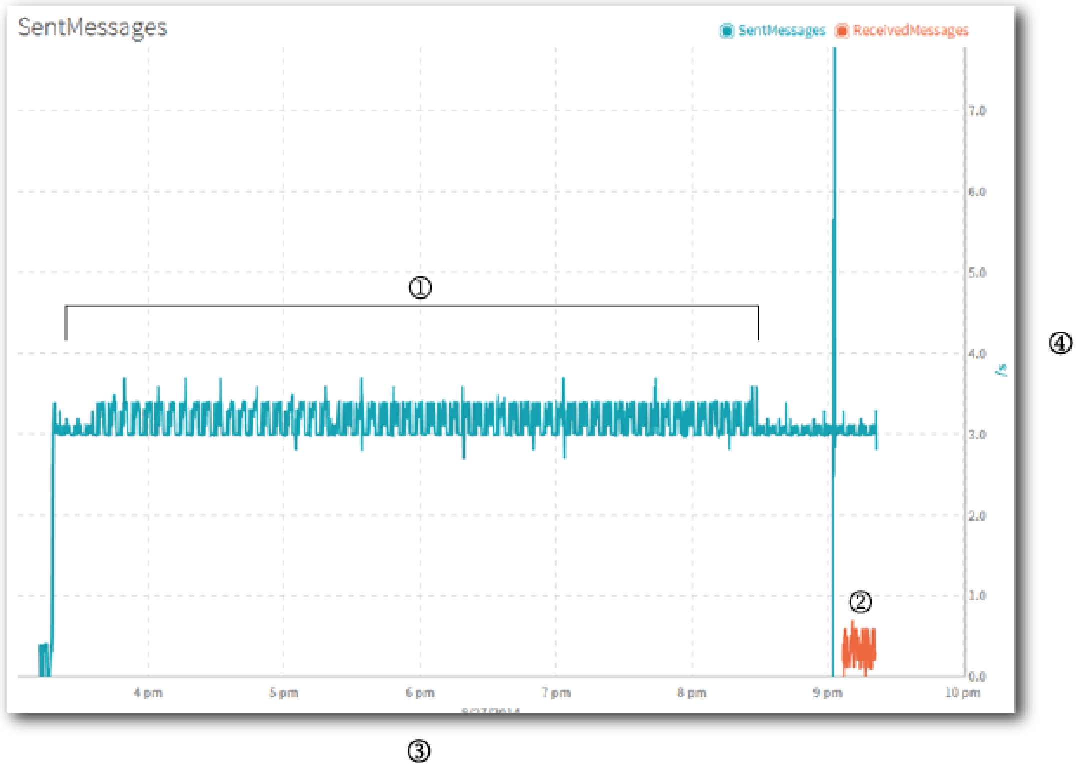
This sequence of events represents only the tip of the iceberg of possibilities. When **`Parse All Properties`** is set to `true`, the **Tokens** component exposes all of the current properties and any that may be added in the future.

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.

Figure 114 *Example of a sent and received messages graph*



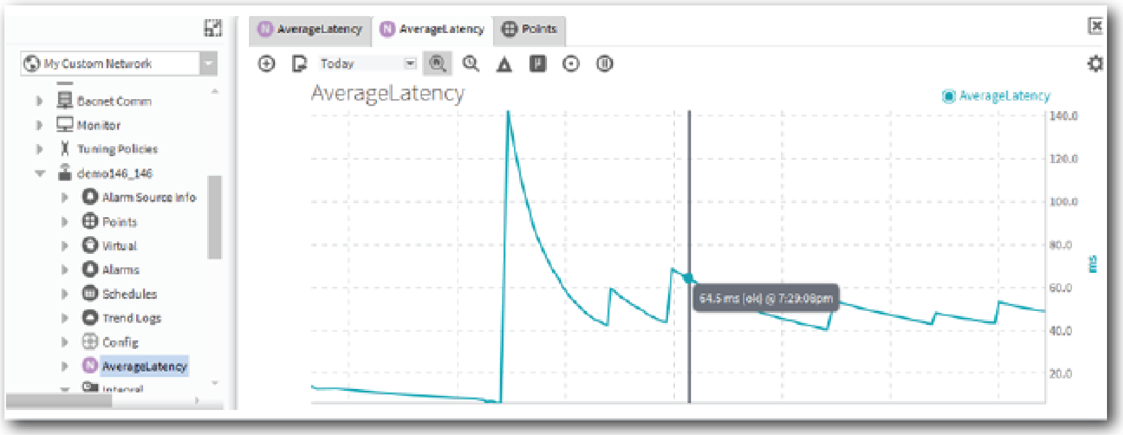
1. Sent message count
2. Received message count
3. Time
4. Per seconds

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.

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).

Figure 115 Average latency chart



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.

Setting up a wiretap

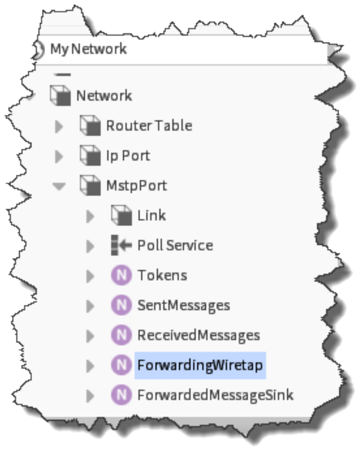
Wiretaps evaluate (sniff) messages after they have been processed by a network port. You add wiretaps as children of the **NetworkPorts** container in the same way that you add metric components.

Prerequisites: Your BACnet network has been set up and all devices are configured and communicating.

Step 1 To prepare the Nav tree, expand **Config→Drivers→BacnetNetwork→Bacnet Comm→Network**.

Step 2 To set up an MS/TP port, open the **bacnet** (Niagara BACnet Driver) palette and expand **NetworkPorts**.

Step 3 Drag the **MstpPort** container to the **Network** container in the Nav tree.



Step 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.

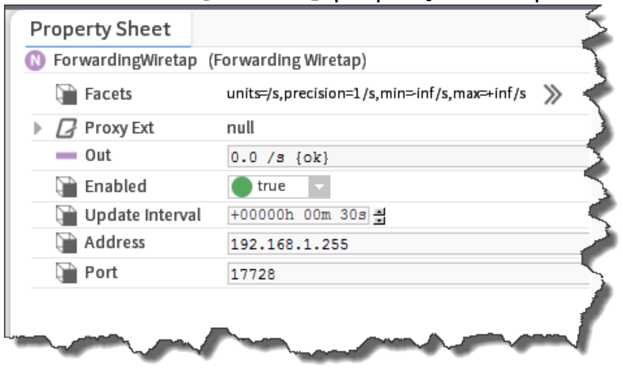
Processing a forwarded wiretap

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.

Prerequisites: You have added the ForwardingWiretap component to the MstpPort container in the Nav tree. You are an experienced Wireshark user.

Step 1 To configure the **`ForwardingWiretap`** component, double-click it in the Nav tree.

The **`ForwardingWiretap`** property sheet opens.



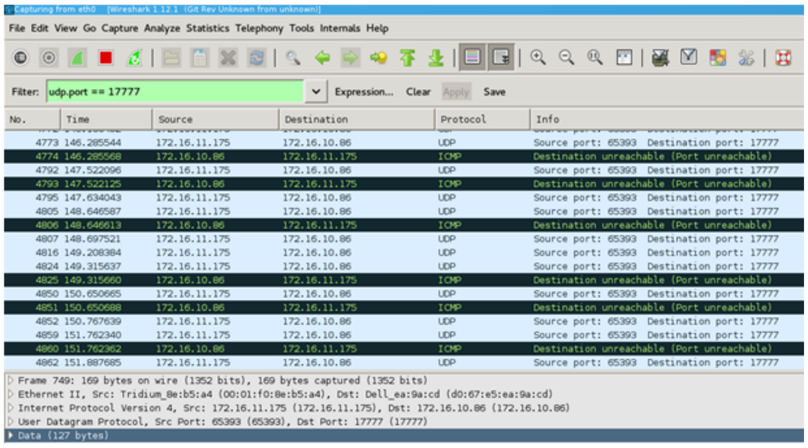
Property Sheet

ForwardingWiretap (Forwarding Wiretap)

Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s
Address	192.168.1.255
Port	17728

Step 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.


Capturing from eth0 [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

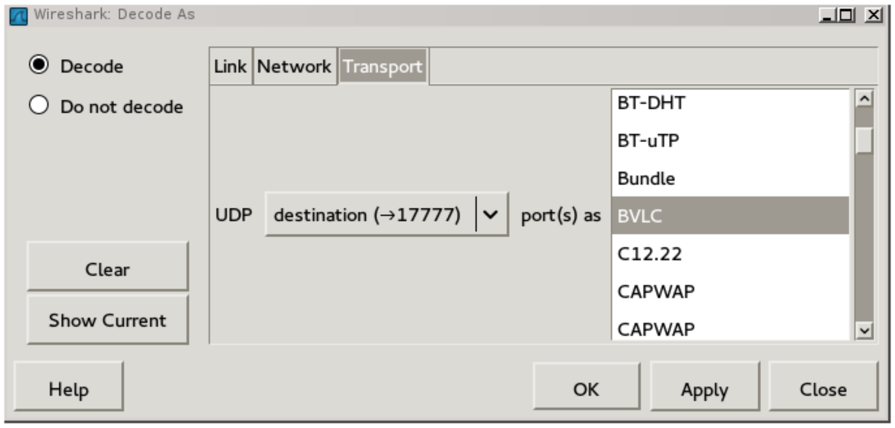
Filter: udp.port == 17777

No.	Time	Source	Destination	Protocol	Info
4773	146.285544	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4774	146.285568	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4792	147.522096	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4793	147.522125	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4795	147.634043	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4805	148.646587	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4806	148.646613	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4807	148.697521	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4816	149.208384	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4824	149.315637	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4825	149.315660	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4850	150.650665	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4851	150.650688	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4852	150.767639	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4859	151.762340	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777
4860	151.762362	172.16.10.86	172.16.11.175	ICMP	Destination unreachable (Port unreachable)
4862	151.887685	172.16.11.175	172.16.10.86	UDP	Source port: 65393 Destination port: 17777

```
▷ Frame 749: 169 bytes on wire (1352 bits), 169 bytes captured (1352 bits)
▷ Ethernet II, Src: Tridium_8e:b5:a4 (00:01:f0:8e:b5:a4), Dst: Dell_ea:9a:cd (d0:67:e5:ea:9a:cd)
▷ Internet Protocol Version 4, Src: 172.16.11.175 (172.16.11.175), Dst: 172.16.10.86 (172.16.10.86)
▷ User Datagram Protocol, Src Port: 65393 (65393), Dst Port: 17777 (17777)
▶ Data (127 bytes)
```

Step 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.



Wireshark: Decode As

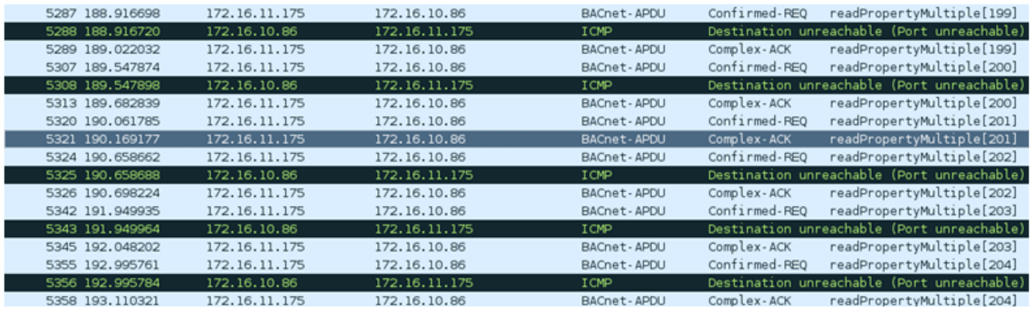
[✓ Decode] Do not decode

Link | Network | Transport

UDP destination (→17777) port(s) as BVLC

Step 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]
5308 189.547898   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 191.949964   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.”

Step 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.

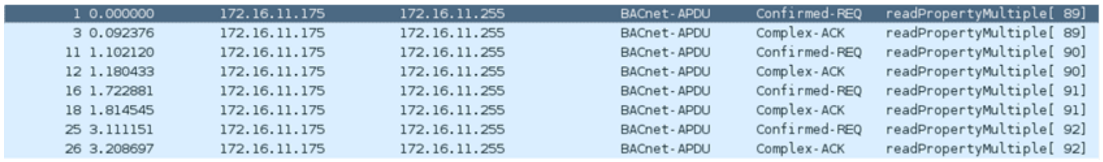
c. Set up a process to listen for these incoming messages and discard them.

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 form the MS/TP trunk:



```
  1 0.000000   172.16.11.175   172.16.11.255   BACnet-APDU   Confirmed-REQ   readPropertyMultiple[ 89]
  3 0.092376   172.16.11.175   172.16.11.255   BACnet-APDU   Complex-ACK     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]
```

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.

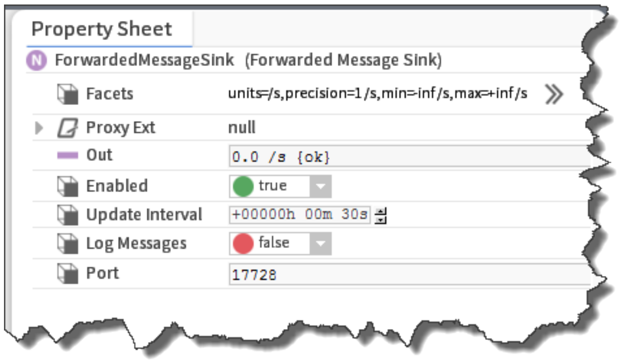
Prerequisites: You have added the ForwardedMessageSink component to the MstpPort container in the Nav tree.

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.

Step 1 To configure the port, double-click the **`ForwardMessageSink`**.

The **`ForwardMessageSink`** property sheet opens.


Property Sheet
ForwardedMessageSink (Forwarded Message Sink)

Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s
Log Messages	false
Port	17728

Step 2 Set the **`Port`** to the same value used by the forwarder on the controller.

Step 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.

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.

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.

Reference

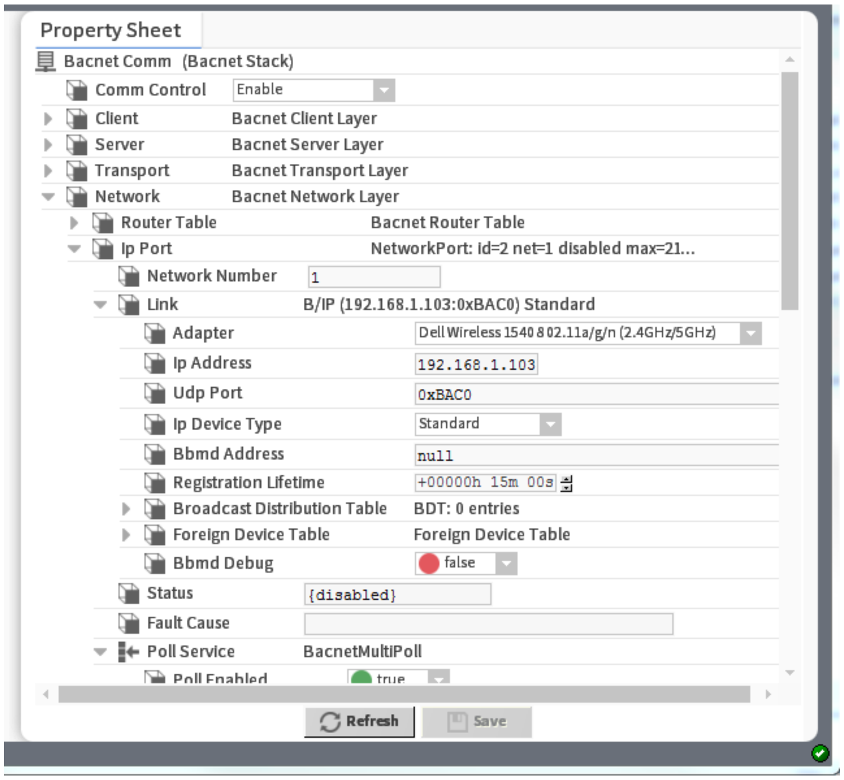
The topics that follow provide detailed documentation for each component and plugin that supports this system feature.

Properties

Network

The **Network** container under **BacnetComm** determines the BACnet network-layer configuration for the station. You access **BacnetComm** directly in the Nav tree.

Figure 116 BACnet network properties



Bacnet Comm (Bacnet Stack)	
Comm Control	Enable
Client	Bacnet Client Layer
Server	Bacnet Server Layer
Transport	Bacnet Transport Layer
Network	Bacnet Network Layer
Router Table	Bacnet Router Table
Ip Port	NetworkPort: id=2 net=1 disabled max=21...
Network Number	1
Link	B/IP (192.168.1.103:0xBAC0) Standard
Adapter	Dell Wireless 1540 802.11a/g/n (2.4GHz/5GHz)
Ip Address	192.168.1.103
Udp Port	0xBAC0
Ip Device Type	Standard
Bbmd Address	null
Registration Lifetime	+00000h 15m 00s
Broadcast Distribution Table	BDT: 0 entries
Foreign Device Table	Foreign Device Table
Bbmd Debug	false
Status	{disabled}
Fault Cause	
Poll Service	BacnetMultiPoll
Poll Enabled	true

Property	Value	Description
Router Table		
Ip Port		See Network properties, Ip Port, page 130.
Routing Enabled	`true` (default) or `false`	
Maintain Routing Enabled	`true` or `false` (default)	
Minimum router Update time	milliseconds (default: 500)	
Router Discovery Timeout	milliseconds (default: 5000)	

Property	Value	Description
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	`hours minutes seconds`	
Link, Broadcast Distribution Table	BDT: 0 entries	
Link, Foreign Device Table		
Link, Bbmd Debug	`true` or `false` (default)	
Status [component]	text	Read-only field. Indicates the condition of the component at last polling. • `{ok}` indicates that the component is polling successfully. • `{down}` indicates that polling is unsuccessful, perhaps because of an incorrect property. • `{disabled}` indicates that the **`Enable`** property is set to `false`. • `fault` indicates another problem.
Fault Cause	text	Read-only field. Indicates why the network, component, or extension is in fault.
Poll Service		See Network properties, Ip Port, Poll Service, page 131.
Max Devices	max	
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.

Property	Value	Description
Port Id		
Port Info		

Network properties, Ip Port, Poll Service

Property	Value	Description
Poll enabled	`true` (default) or `false`	
Fast Rate	`hours minutes seconds`	
Normal Rate	`hours minutes seconds`	
Slow Rate	`hours minutes seconds`	
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	

Property	Value	Description
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 [component]	text	Read-only field. Indicates the condition of the component at last polling. • `{ok}` indicates that the component is polling successfully. • `{down}` indicates that polling is unsuccessful, perhaps because of an incorrect property. • `{disabled}` indicates that the **`Enable`** property is set to `false`. • `fault` indicates another problem.
Fault Cause	text	Read-only field. Indicates why the network, component, or extension is in fault.
Poll Service		See Network properties, Ip Port, Poll Service, page 131
Max Devices	max	
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.
Port Id		
Port Info		

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**.

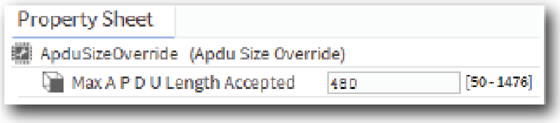
bacnetUtil components

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 117 ApduSizeOverride property



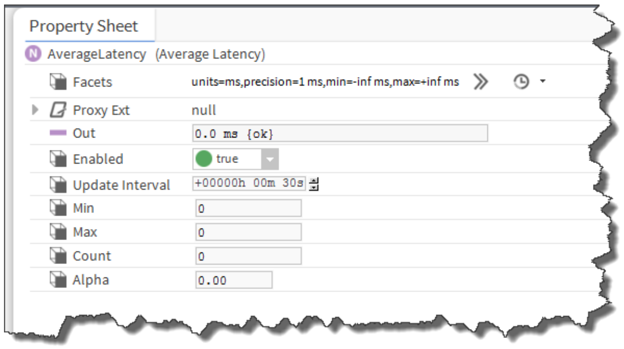
Property Sheet
ApduSizeOverride (Apdu Size Override)
Max A P D U Length Accepted 480 [50 - 1476]

Property	Value	Description
Max A P D U Length Accepted	number from 50 to 1476	The size of the APDU.

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 118 Average Latency properties



Property Sheet
AverageLatency (Average Latency)
Facets units=ms,precision=1 ms,min=-inf ms,max=+inf ms
Proxy Ext null
Out 0.0 ms {ok}
Enabled true
Update Interval +00000h 00m 30s
Min 0
Max 0
Count 0
Alpha 0.00

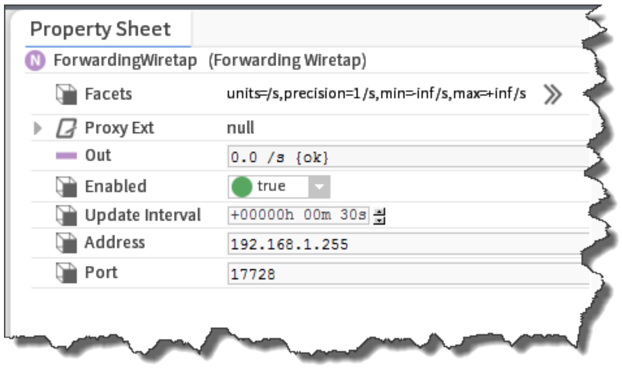
Property	Value	Description
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display “ON” and the facet falseText to display “OFF.” "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point’s default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a	A frozen property found on each point that indicates from where the point’s data originate, including details specific to the parentage of the point’s network and communications (driver).

Property	Value	Description
	BooleanWritable. The point under the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.
Update Interval		
Min		
Max		
Count		
Alpha		

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 119 ForwardingWiretap properties


Property Sheet
ForwardingWiretap (Forwarding Wiretap)

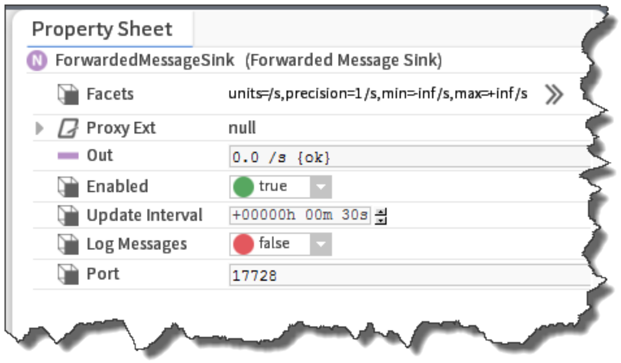
Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s
Address	192.168.1.255
Port	17728

Property	Value	Description
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display "ON" and the facet falseText to display "OFF." "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point's default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a BooleanWritable. The point under the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	A frozen property found on each point that indicates from where the point's data originate, including details specific to the parentage of the point's network and communications (driver).
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.
Update Interval	hours minutes seconds	Indicates how frequently new information is reported.
Address	IP address	The IP address of the source or destination device.
Port	numeric	The port number on the controller or computer.

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 120 ForwardedMessageSink properties



Property Sheet

ForwardedMessageSink (Forwarded Message Sink)

Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s
Log Messages	false
Port	17728

Property	**Value**	**Description**
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display “ON” and the facet falseText to display “OFF.” "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point’s default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a BooleanWritable. The point under the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	A frozen property found on each point that indicates from where the point’s data originate, including details specific to the parentage of the point’s network and communications (driver).
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.

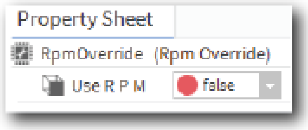
Property	Value	Description
Update Interval	hours minutes seconds	Indicates how frequently new information is reported.
Log Messages	`true` or `false`	When set to `true`, the system creates a hex dump of each received message and stores it in station output. Setting this property to `false` disables the creation of a hex dump of each received message.
Port	numeric	The port number on the controller or computer.

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 ablel 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 121 RpmOverride property



RpmOverride (Rpm Override)
Use R P M: false

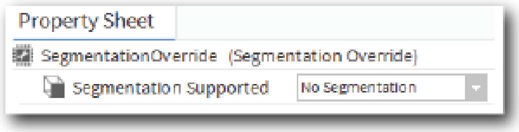
Property	Value	Description
Use R P M	`true` or `false`	`true` sends RPM messages to the device. This is the case even if the device does not support these messages. Expect many unsupported service errors to be returned by the device. `false` causes the system to ignore any claimed support for RPM. The system only sends **`readProperty`** messages to the device, even if the device supports **`readPropertyMultiple`**.

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 122 SegmentationOverride property



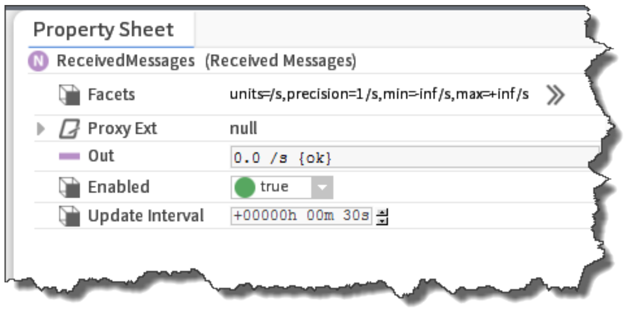
SegmentationOverride (Segmentation Override)
Segmentation Supported: No Segmentation

Property	Value	Description
Segmentation Supported	drop-down list	

SentMessages and 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 123 *NetworkPort properties*



Property Sheet

ReceivedMessages (Received Messages)

Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s

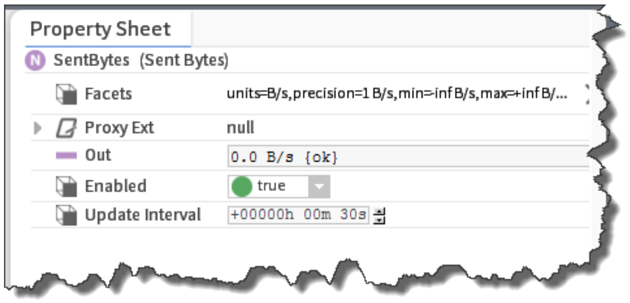
Property	Value	Description
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display “ON” and the facet falseText to display “OFF.” "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point’s default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a BooleanWritable. The point under the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	A frozen property found on each point that indicates from where the point’s data originate, including details specific to the parentage of the point’s network and communications (driver).

Property	Value	Description
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Update Interval	hours minutes seconds	Indicates how frequently new information is reported.

SentBytes and 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 124 *SentBytes properties*


Property Sheet
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 +00000h 00m 30s

Property	Value	Description
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display “ON” and the facet falseText to display “OFF.” "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point’s default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a BooleanWritable. The point under	A frozen property found on each point that indicates from where the point’s data originate, including details specific to the parentage of the point’s network and communications (driver).

Property	Value	Description
	the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Update Interval	hours minutes seconds	Indicates how frequently new information is reported.

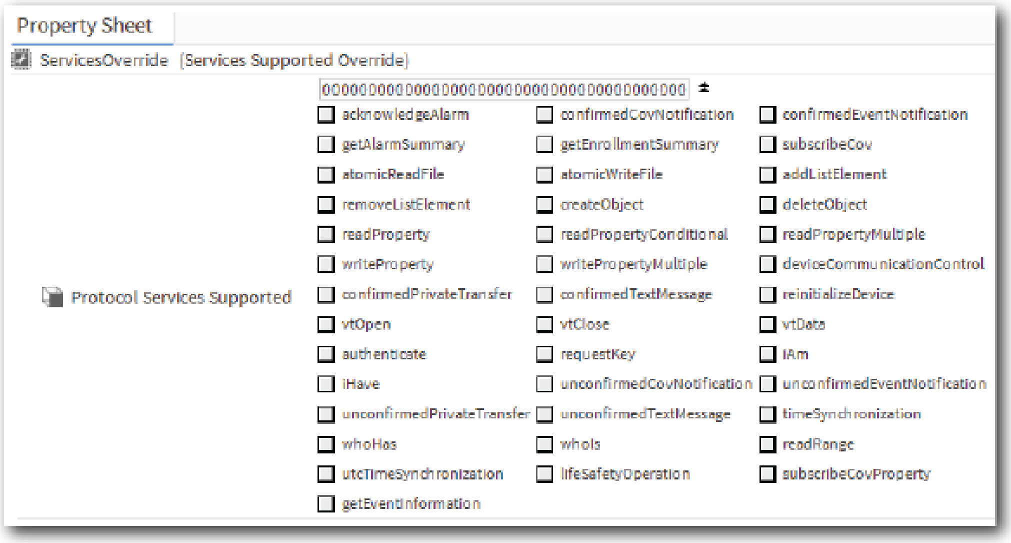
ServicesOverride

When added to a BBacnetDevice, 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 125 *ServicesOverride properties*



Property Sheet

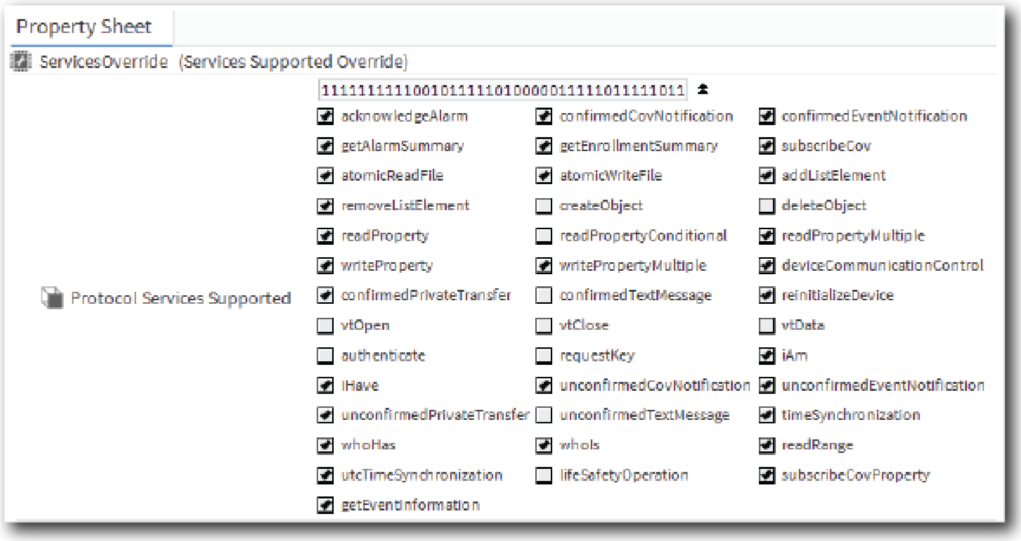
ServicesOverride (Services Supported Override)

Protocol Services Supported 0000000000000000000000000000000000000000

☐ acknowledgeAlarm	☐ confirmedCovNotification	☐ confirmedEventNotification
☐ getAlarmSummary	☐ getEnrollmentSummary	☐ subscribeCov
☐ atomicReadFile	☐ atomicWriteFile	☐ addListElement
☐ removeListElement	☐ createObject	☐ deleteObject
☐ readProperty	☐ readPropertyConditional	☐ readPropertyMultiple
☐ writeProperty	☐ writePropertyMultiple	☐ deviceCommunicationControl
☐ confirmedPrivateTransfer	☐ confirmedTextMessage	☐ reinitializeDevice
☐ vtOpen	☐ vtClose	☐ vtData
☐ authenticate	☐ requestKey	☐ IAm
☐ iHave	☐ unconfirmedCovNotification	☐ unconfirmedEventNotification
☐ unconfirmedPrivateTransfer	☐ unconfirmedTextMessage	☐ timeSynchronization
☐ whoHas	☐ whoIs	☐ readRange
☐ utcTimeSynchronization	☐ lifeSafetyOperation	☐ subscribeCovProperty
☐ getEventInformation		

As a child of a BBacnetDevice, 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 126 ServicesOverride example



Property Sheet

ServicesOverride (Services Supported Override)

Protocol Services Supported

1111111111001011111010000011111011111011

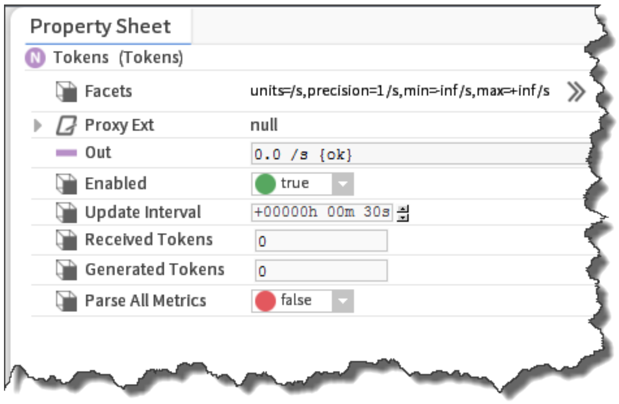
✓ acknowledgeAlarm	✓ confirmedCovNotification	✓ confirmedEventNotification
✓ getAlarmSummary	✓ getEnrollmentSummary	✓ subscribeCov
✓ atomicReadFile	✓ atomicWriteFile	✓ addListElement
✓ removeListElement	☐ createObject	☐ deleteObject
✓ readProperty	☐ readPropertyConditional	✓ readPropertyMultiple
✓ writeProperty	✓ writePropertyMultiple	✓ deviceCommunicationControl
✓ confirmedPrivateTransfer	☐ confirmedTextMessage	✓ reinitializeDevice
☐ vtOpen	☐ vtClose	☐ vtData
☐ authenticate	☐ requestKey	✓ iAm
✓ iHave	✓ unconfirmedCovNotification	✓ unconfirmedEventNotification
✓ unconfirmedPrivateTransfer	☐ unconfirmedTextMessage	✓ timeSynchronization
✓ whoHas	✓ whoIs	✓ readRange
✓ utcTimeSynchronization	☐ lifeSafetyOperation	✓ subscribeCovProperty
✓ getEventInformation		

Once you select or clear each protocol and save, Niagara uses the overrides in preference to device-provided values.

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.

Figure 127 Tokens properties



Property Sheet

Tokens (Tokens)

Facets	units=/s,precision=1/s,min=-inf/s,max=+inf/s
Proxy Ext	null
Out	0.0 /s {ok}
Enabled	true
Update Interval	+00000h 00m 30s
Received Tokens	0
Generated Tokens	0
Parse All Metrics	false

Type	Value	Description
Facets	*trueText* (default) or *falseText*	Facets contain additional data applied to input and output values. • *trueText* is the text to display when output is true • *falseText* is the text to display when output is false. For example, you might want to set the facet trueText to display "ON" and the facet falseText to display "OFF." "Units of measurement" is also a type of facet. You can view Facets on the Slot sheet and edit them from a component Property sheet by clicking the >> icon to display the **Config Facets** window.
Proxy ext	`null` indicates that the point is an empty placeholder. The station itself originates the point's default Out value.`driver type` identifies the driver and point type. For example, a proxy ext of BacnetBoolean identifies a BooleanWritable. The point under the **Points** container of a Bacnet Device has a proxy extension of BacnetBooleanProxyExt.	A frozen property found on each point that indicates from where the point's data originate, including details specific to the parentage of the point's network and communications (driver).
Out	`value facets` {ok}	The current out value, including any point facets. Out display of a proxy point defaults to only the single (configured) property value, along with Niagara status for the proxy point. You can edit point facets to poll for additional properties, such as the native "statusFlags" and/or "priorityArray" level.
Enabled [general]	`true` or `false`	Activates and deactivates use of the function.
Update Interval	hours minutes seconds	Indicates how frequently new information is reported.
Mstp Os Name	/dev/*name*	Where *name* is mstp followed by a single digit to identify the MS/TP port. This is the order in which the ports are started (the order in which they appear under the BBacnetNetworkLayer). For example, `mstp1`, `mstp2`.
Received Tokens	number	
Generated Tokens	number	
Parse All Metrics	true (default) or false	
serialport	/dev/ser*number*	Where *number* is a single digit that identifies the serial port.

Type	Value	Description
baudrate	0.00	
macaddress	0.00	
maxinfoframes	0.00	
maxmaster	0.00	
master_state	0.00	
receive_state	0.00	
nextstation	00.00	
pollstation	0.00	
solemaster	`true` or `false` (default)	
declaresolemaster	0.00	
framecount	0.00	
tokencount	0.00	
silencetimer	0.00	
n40bitdelay	0.00	
chartimens	0.00	
validframes	0.00	
invalidframes	0.00	
apptx	0.00	
apprx	0.00	
skippedframes	0.00	
unwantedframes	0.00	
testrequest_rx	0.00	
testrequest_tx	0.00	
testresponse_rx	0.00	
testresponse_tx	0.00	
extframes_rx	0.00	
extframes_tx	0.00	
ip_6lobac_rx	0.00	
ip_6lobac_tx	0.00	
n6lobac_rx_brdcst	0.00	
n6lobac_tx_brdcst	0.00	
badheadercrc	0.00	
baddatacrc16	0.00	
baddatacrc32	0.00	

Type	Value	Description
txq_fullcount	0.00	
serialtxfails	0.00	