

# Technical Document

## NiagaraAX-3.7 and AX-3.8 Graphics Guide

November 8, 2013



# NiagaraAX-3.7 and AX-3.8 Graphics Guide

## Legal Notices

### Confidentiality notice

The information contained in this document is confidential information of Tridium, Inc., a Delaware corporation ("Tridium"). Such information and the software described herein, is furnished under a license agreement and may be used only in accordance with that agreement.

The information contained in this document is provided solely for use by Tridium employees, licensees, and system owners; and, except as permitted under the below copyright notice, is not to be released to, or reproduced for, anyone else.

While every effort has been made to assure the accuracy of this document, Tridium is not responsible for damages of any kind, including without limitation consequential damages, arising from the application of the information contained herein. Information and specifications published here are current as of the date of this publication and are subject to change without notice. The latest product specifications can be found by contacting our corporate headquarters, Richmond, Virginia.

### Trademark notice

BACnet and ASHRAE are registered trademarks of American Society of Heating, Refrigerating and Air-Conditioning Engineers. Microsoft, Excel, Internet Explorer, Windows, Windows Vista, Windows Server, and SQL Server are registered trademarks of Microsoft Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Mozilla and Firefox are trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, NiagaraAX Framework, and Sedona Framework are registered trademarks, and Workbench, WorkPlaceAX, and AXSupervisor, are trademarks of Tridium Inc. All other product names and services mentioned in this publication that is known to be trademarks, registered trademarks, or service marks are the property of their respective owners.

### Copyright and patent notice

This document may be copied by parties who are authorized to distribute Tridium products in connection with distribution of those products, subject to the contracts that authorize such distribution. It may not otherwise, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Tridium, Inc.

© Tridium, Inc. 2013. All rights reserved. The product(s) described herein may be covered by one or more U.S. or foreign patents of Tridium.

# PREFACE

## About this guide

This document provides information on the graphical presentation tools that are available in the NiagaraAX-3.7-3.8 Workbench.

Included in the guide are basic procedures for creating and configuring graphics in the Workbench Px Editor, as well as a Px graphics reference which describes presentation concepts and architecture principles.

- To begin creating and configuring graphics in the Workbench Px Editor, see [Chapter 2 Common graphics tasks, page 3](#)
- For detailed information on the basic presentation concepts and architecture principles, see [Chapter 3 Px graphics reference, page 49](#)

### **Change log**

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

- Initial publication: November 8, 2013

### **Related documents**

Additional information on the NiagaraAX system and Niagara Workbench is available in the following document.

- *NiagaraAX User Guide* (<module://docUser/doc/index.html>)



# CONTENTS

<b>Confidentiality notice .....</b>	<b>2</b>
<b>Trademark notice .....</b>	<b>2</b>
<b>Copyright and patent notice .....</b>	<b>2</b>
<b>About this guide .....</b>	<b>iii</b>
Change log .....	iii
Related documents.....	iii
<b>Chapter 1 About Presentation XML (Px) .....</b>	<b>1</b>
What's new in AX-3.8? .....	2
<b>Chapter 2 Common graphics tasks .....</b>	<b>3</b>
<b>Add a comments box on Px views .....</b>	<b>4</b>
Add and configure components for a comment box.....	4
Add a text entry field and save button.....	4
Embed a History Table in your Px view.....	5
<b>Add data bindings.....</b>	<b>6</b>
<b>Add hyperlinks using Popup Bindings (an example) .....</b>	<b>6</b>
<b>Add Px Views .....</b>	<b>7</b>
<b>Add widgets using the Make Widget wizard .....</b>	<b>8</b>
<b>Add WeatherReports on Px Views .....</b>	<b>8</b>
<b>Animate a widget property.....</b>	<b>9</b>
<b>Animate using static SVG images .....</b>	<b>10</b>
<b>Create a “global” navigation menu using PxInclude .....</b>	<b>11</b>
<b>Create a scalable image using the Picture widget .....</b>	<b>12</b>
<b>Embed a history chart in a Px view .....</b>	<b>13</b>
<b>Embed Px files with ORD variables using PxInclude.....</b>	<b>13</b>
<b>Group objects using Px Layers .....</b>	<b>14</b>
Add a Px Layer.....	15
Assign/Unassign objects to a layer.....	15
Rename a Px Layer.....	15
Delete a Px Layer.....	15
<b>Launch the History Chart Builder in a Px view .....</b>	<b>16</b>
<b>Make Bound Label widgets .....</b>	<b>16</b>
<b>Make Chart widgets.....</b>	<b>20</b>
<b>Make component Properties-based widgets.....</b>	<b>23</b>
<b>Make Palette kit-based widgets.....</b>	<b>25</b>
<b>Make Time Plot widgets.....</b>	<b>28</b>
<b>Make Workbench view-based widgets .....</b>	<b>31</b>
<b>Make Action widgets .....</b>	<b>34</b>
<b>Relativize absolute ORDs .....</b>	<b>35</b>

- Use the zooming options .....36**
  - Zoom-in on a Px Page ..... 36
  - Zoom-out on a Px Page ..... 37
  - Reset zoom on Px Page ..... 37
- Workflow for using the PxInclude Widget and ORD Variables .....37**
  - Using a PxInclude Widget with a single ORD variable (an example) ..... 38
  - Using PxInclude Widgets with multiple ORD variables (an example) ..... 39
- Use the visible property in Hx profile (an example) .....39**
- Customizing the login screen .....41**
- Apply visual styles to Px views .....42**
  - Using Px Properties ..... 42
- Build navigation files.....42**
  - Creating new nav files..... 43
  - Deleting nav files..... 43
  - Renaming nav files..... 43
  - Editing nav files..... 43
    - Open nav files ..... 43
    - Add nodes in nav files..... 44
    - Delete nodes in nav files..... 44
    - Edit node hierarchy ..... 44
    - Edit node properties..... 45
- Generate reports.....45**
  - Reporting process workflow ..... 45
  - Set up the reporting service..... 46
  - Set up report data in a ComponentGrid ..... 46
  - Creating A ReportPxFile..... 46
  - Configuring ExportSource component..... 47
  - Configuring EmailRecipient component ..... 47

**Chapter 3 Px graphics reference ..... 49**

- About Px views .....49**
  - Px views as slot properties..... 50
  - About the Px View wizard ..... 50
  - Choosing a Px Media type..... 51
- About Px files.....53**
  - Shared Px files ..... 54
- About Px viewer .....55**
- About Px Editor.....55**
  - About Px Editor Canvas ..... 56
    - About alignment and distribution tools..... 57
  - About Widget sizing in Px Editor ..... 58
  - About Px Properties ..... 58
    - Px Property Concepts ..... 59
    - About the Px Property palette ..... 60
  - About Px Layers ..... 61
    - About the Px Layers palette..... 62
  - About SVG support ..... 62
    - About SVG rendering in Workbench..... 63
    - AX-compatible SVG images ..... 63
  - About Zoom capability in Px Editor..... 63
- About the View Source XML dialog box .....64**
- About Widgets .....65**
  - Widget layout..... 65
  - Types of panes..... 66
  - Widget painting ..... 68
  - Widget commands..... 68
  - Widget properties..... 69
  - About the kitPxGraphics palette..... 69

About third-party graphics collections ..... 71

**About developing for portability.....71**

**About the PxInclude Widget .....76**

    PxInclude Widget concepts ..... 76

    Types of PxInclude Widget properties..... 78

    Use PxInclude Widgets ..... 79

    About ORD Variables ..... 80

**About Animating Graphics.....81**

    Data binding ..... 81

    Types of data bindings..... 83

    Types of binding properties..... 84

    About bound label bindings ..... 86

    About value bindings..... 87

        Types of Converters..... 87

    About spectrum bindings ..... 88

    About setpoint bindings ..... 88

    About increment setpoint bindings ..... 89

    About spectrum setpoint bindings ..... 89

    About action bindings..... 90

    About table bindings..... 90

    About field editor bindings..... 90

    About Popup Bindings ..... 91

    About relative and absolute bindings..... 91

**About types of Px target media.....92**

**About profiles.....93**

    About Web Profiles ..... 94

        Default Wb Web Profile ..... 94

        Simple Admin Wb Web Profile..... 95

        Basic Wb Web Profile ..... 96

        Default Mobile Web Profile..... 97

        Default Hx Profile..... 97

        Basic Hx Profile ..... 99

        Basic Touchscreen Profile..... 100

        Default Touchscreen Profile ..... 101

        Handheld Touchscreen Profile..... 101

    About Kiosk Profiles ..... 101

        Basic Kiosk Profile ..... 102

        Default Kiosk Profile ..... 102

        Handheld Kiosk Profile ..... 102

**About the Px Editor workflow ..... 102**

**About the Make Widget wizard..... 102**

**About the Nav file ..... 104**

    About Nav file elements ..... 105

**About the Nav File Editor..... 106**

    About Nav File Editor source objects ..... 107

    About Nav File Editor result tree ..... 108

    About Nav File Editor buttons ..... 109

**About Reporting ..... 109**

    Types of report components..... 110

        About the ReportService component ..... 111

        About the ExportSource component ..... 111

        About the EmailRecipient component ..... 112

        About the FileRecipient component..... 113

        About the BqlGrid component ..... 113

        About the ComponentGrid component ..... 114

        About the ReportPane component ..... 115

        About the SectionHeader component ..... 116

    About the Grid Table view..... 116

    About the Grid Label Pane view..... 117

    About the Grid Editor view ..... 119

About using the Grid Editor view ..... 120  
 About editing rows/columns in a ComponentGrid ..... 121  
 About the Report Edit Column/Edit Row dialog boxes ..... 121  
 About the ReportPxFile ..... 122

**About Mobile ..... 122**  
 About Mobile Px App..... 123

**Chapter 4 Component Guides ..... 125**

**Components in bajai module ..... 125**  
 BorderPane..... 125  
 Bookmark..... 125  
 BoundTable ..... 125  
 Button..... 125  
 CanvasPane ..... 125  
 CheckBox ..... 126  
 ConstrainedPane ..... 126  
 EdgePane ..... 126  
 Ellipse ..... 126  
 ExpandablePane..... 126  
 FlowPane ..... 126  
 GridPane ..... 126  
 HyperlinkLabel..... 127  
 Label..... 127  
 Line ..... 127  
 NullWidget ..... 127  
 Path..... 128  
 Picture..... 128  
 Polygon ..... 128  
 PxInclude ..... 128  
 RadioButton..... 128  
 Rect..... 128  
 ScrollPane ..... 128  
 Separator ..... 129  
 Slider ..... 129  
 SplitPane..... 129  
 TabbedPane..... 129  
 TextEditorOptions..... 129  
 ToggleButton ..... 131  
 ValueBinding..... 131

**Components in chart module ..... 131**  
 BarChart..... 131  
 ChartCanvas..... 131  
 ChartHeader..... 131  
 ChartPane ..... 131  
 DefaultChartLegend..... 131  
 LineChart ..... 131

**Components in gx module ..... 132**  
 Brush ..... 132

**Components in kitPx module ..... 132**  
 ActionBinding..... 132  
 AnalogMeter ..... 132  
 Bargraph ..... 132  
 BackButton ..... 133  
 BoundLabel ..... 133  
 BoundLabelBinding..... 133  
 ButtonGroup ..... 133  
 ButtonGroupBinding ..... 134  
 ExportButton..... 134  
 ForwardButton..... 134  
 GenericFieldEditor ..... 135  
 ImageButton..... 135



---

IncrementSetPointBinding .....	135
LogoffButton.....	136
RefreshButton.....	136
SaveButton .....	136
SetPointBinding .....	137
SetPointFieldEditor .....	137
<b>Components in pxeditor module.....</b>	<b>137</b>
NewPxViewDialog.....	137
<b>Components in report module .....</b>	<b>137</b>
BqlGrid .....	137
ComponentGrid .....	137
EmailRecipient.....	138
ExportSource.....	138
FileRecipient .....	138
ReportPane.....	138
ReportService.....	138
SectionHeader.....	138
<b>Chapter 5 Plugin Guides .....</b>	<b>139</b>
<b>Plugins in pxEditor module.....</b>	<b>139</b>
pxEditor.....	139
<b>Plugins in report module .....</b>	<b>139</b>
GridTable .....	139
GridLabelPane .....	139
ComponentGridEditor.....	140
<b>Index .....</b>	<b>141</b>



# CHAPTER 1 ABOUT PRESENTATION XML (Px)

## TOPICS COVERED IN THIS CHAPTER

- **What's new in AX-3.8?**

This document describes the graphical presentation tools that are available in NiagaraAX Workbench, as well as basic presentation concepts, architecture principles, and basic procedures for creating and configuring graphics in the Workbench Px Editor.

Applications can be assembled with components using the graphical tools in Niagara Workbench. This graphical environment provides tools for you to build new applications and you do not need to be a Java developer to use them.

If you plan to visually display the control logic that you develop in Workbench (for operations or for engineering purposes) then you need to understand the basic principles of Px and understand the capabilities and limitations of your target media. Obviously, graphics and text look different and have different limitations on a hand held display than on a desktop or laptop web browser. It is important to develop Px files with the primary target media types in mind and to test your Px views in the target media types as you develop them.

Usually, it is easier to complete the engineering of your control logic using the wire sheet and other views before beginning to design the presentation of that logic. There are features and tools in the Px Editor, such as the Make Widget wizard, that allow you to drag and drop components from the Nav tree onto various panes in the Px Editor. If you have already built the logic, it should be visible in the Nav tree and available for drag and drop.

The general process of creating presentation views for control logic can follow many different paths. The major steps generally go as follows:

- **Create your view**

When you create a view, you are creating a relationship between a Px file and a component. The Px file defines the view and that view may be associated with one or more components of various types, such as folders and points.

Refer to the following sections for information related to Px views:

- [Chapter 2 Common graphics tasks, page 3](#)
- [About Px views, page 49](#)
- [About Px files, page 53](#)
- [About Px viewer, page 55](#)

- **Bind your data**

After creating a view, add graphic visualizations (called widgets) to the file and pass data to these widgets using data binding. The bound data from the control objects animates and updates the widgets. Refer to the following sections for information related to widgets and data binding:

- [Chapter 2 Common graphics tasks, page 3](#)
- [About Px Editor, page 55](#)
- [About Widgets, page 65](#)
- [About the PxInclude Widget, page 76](#)
- [About Animating Graphics, page 81](#)
- [About types of Px target media, page 92](#)
- [About Web Profiles, page 94](#)

- [About the Make Widget wizard, page 102](#)
- **Create a nav file**

In order to easily find and navigate between views, a special file type (“nav” file) is available for creating a “customizable” navigation tree. Edit the nav file using the Nav File Editor and assign a particular nav file to a user in the user’s profile (using the User Manager view). Refer to the following sections for information related to the nav file:

  - [Chapter 2 Common graphics tasks, page 3](#)
  - [About the Nav file, page 104](#)
  - [About the Nav File Editor, page 106](#)
- **Create and distribute a report**

The Reporting function helps you design, display, and deliver data to online views, printed pages, and distribute via email. Refer to the following sections for information related to Reporting:

  - [Chapter 2 Common graphics tasks, page 3](#)
  - [Generate reports, page 45](#)
  - [About Reporting, page 109](#)

## What's new in AX-3.8?

If you are already familiar with the graphics features of the AX-3.7 Workbench, the following summary of changes may be helpful in using the AX-3.8 version.

- Starting in AX 3.8, Workbench includes SVG rendering capability. With the svgBatik module installed, SVG images can be used in your Px pages just about anywhere a GIF, PNG, or JPG can. An SVG image can be scaled up to larger sizes without degrading (becoming pixelated) as PNG images do. Also, large images scale down in size while retaining image quality and legibility. In order to provide this scaling functionality, a new widget called a Picture has been added to the **bajau** palette.
- In AX 3.8, PxEditor includes zooming capability. You can zoom in and out on Px pages while working in edit mode. Zoom changes the magnification of the page bringing it in closer for detailed work and zooming out to see the whole page.
- In AX 3.8, a number of graphical widgets have been added to the kitPxGraphics module. New widgets in the kitPxGraphics palette include various animated, static, and high-detail graphics in multiple sizes.

For more details see:

- [About SVG support, page 62](#)
- [About Zoom capability in Px Editor, page 63](#)
- [About Widgets, page 65](#)

# CHAPTER 2 COMMON GRAPHICS TASKS

## TOPICS COVERED IN THIS CHAPTER

- Add a comments box on Px views
- Add data bindings
- Add hyperlinks using Popup Bindings (an example)
- Add Px Views
- Add widgets using the Make Widget wizard
- Add WeatherReports on Px Views
- Animate a widget property
- Animate using static SVG images
- Create a “global” navigation menu using PxInclude
- Create a scalable image using the Picture widget
- Embed a history chart in a Px view
- Embed Px files with ORD variables using PxInclude
- Group objects using Px Layers
- Launch the History Chart Builder in a Px view
- Make Bound Label widgets
- Make Chart widgets
- Make component Properties-based widgets
- Make Palette kit-based widgets
- Make Time Plot widgets
- Make Workbench view-based widgets
- Make Action widgets
- Relativize absolute ORDs
- Use the zooming options
- Workflow for using the PxInclude Widget and ORD Variables
- Use the visible property in Hx profile (an example)
- Customizing the login screen
- Apply visual styles to Px views
- Build navigation files
- Generate reports

This section provides instructions for and examples of common graphics tasks such as: adding views, making widgets and adding data bindings, as well as creating nav files and generating reports.

A typical workflow for creating a dynamic visual representation of your control logic involves these basic phases:

- **Add a view**

When you create a view, this creates a relationship between a Px file and a component.

- **Bind the data**

Add widgets to your Px file and pass data to these widgets using data binding. Bound data from the control objects animates and updates the widgets.

- **Create a nav file**

Create a navigation tree to easily find and navigate between views.

- **Generate a report**

Optionally, data presented in a view can be converted to a PDF report. You can configure the email recipients and a delivery schedule for reports.

## Add a comments box on Px views

There are a number of ways to accomplish this. One method is described below.

You can provide a comments box on a Px view for users to enter comments or notes. All saved entries are displayed in the Px view as well. The basic workflow for this is as follows:

### **Add and configure components for a comment box**


Part one of adding a comment box to a Px view is to add and configure components in your station.

#### **Prerequisites:**

- An open connection to your station

### **Add and configure the components for a comment box**

---

- Step 1. In the Px Editor Palette side bar, open the **control** palette, and drag and drop a **StringWritable** point to a location in the **Config** node on your station.
- Step 2. In the Palette side bar, open the **history** palette, expand the Extensions folder and drag and drop a **StringCOV** history extension onto the **StringWritable** point added earlier.
- Step 3. In the Nav tree, double-click on the added **StringCov** component to open its Property Sheet view and set the **Enabled** property value to **true**.
- Step 4. In the Nav tree, right-click on the **StringWritable** point and select **Actions > Set** from the popup menu.
- Step 5. In the **Set** dialog box, type some text to generate the first history record, for example: `First text entry.`
- Step 6. In the Nav tree, double-click on the **StringWritable** point to open its Property Sheet view.
- Step 7. In the **Facets** slot, click on the configuration icon ( » ).  
The **Config Facets** dialog box appears.
- Step 8. In the **Config Facets** dialog box, click the **Add** button (  ) and configure the facet key as follows:
  - Key: **multiLine**
  - Type: **Boolean**
  - Value: **true**

In part two of this procedure you will [see page 4](#)

### **Add a text entry field and save button**

Part two of adding a comment box to a Px view, is adding the necessary widgets to display the text entry field and a save button.

#### **Prerequisites:**

- You have completed part one, [Add and configure components for a comment box, page 4](#).

### **Add the widgets for a text entry field and a save button**

---

- Step 1. In the Nav tree, drag and drop your **StringWritable** point onto the canvas in your Px page.

Step 2. In the **Make Widget wizard**, do the following:

- Select **From Palette**
- Open the **kitPx** palette
- Select a **SetPointFieldEditor** widget
- Click **OK**

Step 3. In the PxEditor canvas, double-click on the **SetPointFieldEditor** widget to open the Properties dialog box and configure the **Set Point Binding** ord property to the **in16** slot, and click **OK**.

The text entry field displays on the canvas.

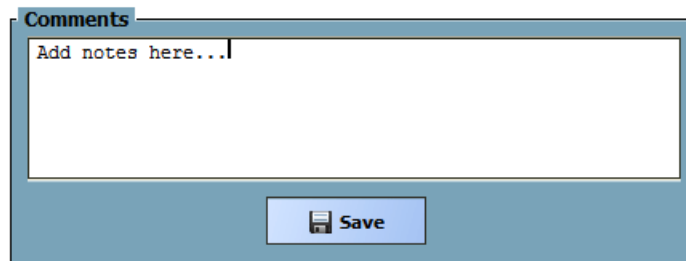
Step 4. In the Palette side bar, open the **kitPx** palette and drag and drop a **Save button** widget to a position near the text entry field on the canvas.

Step 5. Design your comments box as desired, for example, you could enclose the comments box and save button in a Border pane and add a Label as shown below.

Step 6. Click the **Save** icon in the Workbench toolbar to save your changes.

Step 7. Switch to View mode, type some text in the text entry field.

Entering text will activate the **Save** button. Your comments box should resemble the one shown here:



Step 8. Click the **Save** button

Step 9. In the Nav tree, expand the station's **History** node to locate the **StringWritable** history and double-click to open the **History Table** view.

PxTest/StringWritable			2 records
Timestamp	Trend Flags	Status	Value
30-Oct-13 3:01:58 PM EDT	{start}	{ok}	First text entry
30-Oct-13 3:32:30 PM EDT	{}	{ok}	Add notes here...

The history table contains a record of each saved text entry. In the next procedure, you configure your Px view to display the History Table view.

### ***Embed a History Table in your Px view***

Part three of adding a comment box to a Px view, is embedding the History Table view in your Px page to display the saved comments.

#### **Prerequisites:**

- You have completed part one ( [Add and configure components for a comment box, page 4](#)) and part two ( [Add a text entry field and save button, page 4](#)).

### **Add a widget with a Workbench view and History Table**

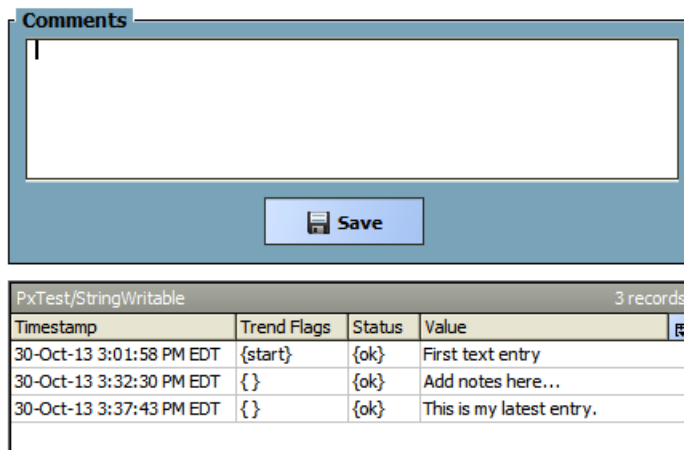
Step 1. Drag the **StringWritable** history component to the Px view.

Step 2. In the Make Widget wizard, select **Workbench view** and **History Table**.

Step 3. In the properties area, set the following values:

- **defaultLiveUpdates** = true
- **show time range editor** = false
- **show delta editor** = false
- **show live updates buttons** = false

After saving your changes, the History Table containing all the text entries displays in the Px view. When a new comment is entered and saved in the Px view, the history table automatically updates, displaying the latest entry. Your comments box and history table should resemble those shown here:




## Add data bindings

Add a binding to a widget that is already on the Px Editor canvas by editing properties.

There are different ways to add a binding to a widget. You can add a binding to a widget using the Make Widget wizard or you can edit the widget properties as described here.

### Add a binding to a widget

Step 1. In the PxEditor canvas, select the desired widget.

Step 2. In the Properties side bar, click the **Add Binding** button (.

Step 3. In the Add Binding dialog box, select a binding type from the options list and click **OK**.

The binding type properties are added to the “binding” area at the bottom of the widget property sheet.

Step 4. Under the binding type properties, click the **ORD** property field.

Step 5. In the **ORD** dialog box, type or browse to the value that you want to bind to the widget and click **OK**.

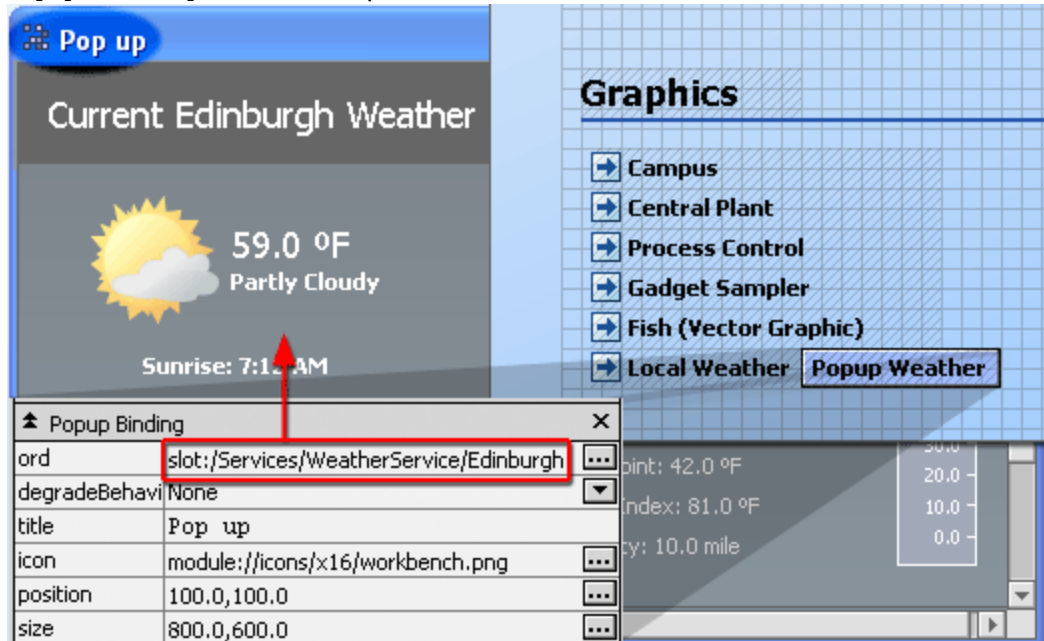
The ORD is added to the binding area of the widget property sheet.

## Add hyperlinks using Popup Bindings (an example)

This example illustrates how to add a hyperlink using a Popup Binding on a Px Button widget.



Typically, a Popup Binding is added to an object that serves as a hyperlink for the Popup window. After adding the Popup Binding, you configure the properties to specify the Px view and Popup Window parameters that you want.



Note the following about this example:

- In the Px Editor view, a Popup Binding is added to a Px button widget (Popup Weather)
- The Popup Binding properties (displayed in the bottom left corner) specify the following:
  - The Popup Window displays the default view of the component at ORD= slot:/Services/WeatherService/Edinburgh
  - No degrade behavior is assigned.
  - The title "Pop up" appears in the top left corner of the Popup Window.
  - The "workbench.png" icon is assigned to the top left corner of the Popup Window.
  - Window X and Y coordinate screen position is x=100 and y=100 pixels.
  - Popup Window size is 800 pixels wide and 600 pixels high.

## Add Px Views

Create a new Px view on a component using the **New Px View** wizard.

When you create a view, you are creating a relationship between a Px file and a component. The Px file defines the view and that view may be associated with one or more components of various types, such as folders and points.

### Add a Px View on a component

Step 1. In the Nav tree, right-click a component and click **Views > New View**

Step 2. In the **New Px View** wizard, enter a **View Name**.

**NOTE:** On completion, the wizard creates a new Px file in the station's Files folder. It assigns a default Px File name (based on the view name) unless you specify a different name. Also, you can assign an existing Px file to the view instead of creating a new one.

Step 3. Click **OK** to finish.

The new Px view displays in the Px Editor. If you choose to have the wizard create a new Px file, it will be a simple default file, as described in [About Px files, page 53](#).

---

**NOTE:** You can also create a new Px view by adding it directly to a slot sheet. Refer to “To add a new slot” in the *NiagaraAX User Guide*, for details about adding new slots.

---

## Add widgets using the Make Widget wizard

The Make Widget wizard dialog box provides fields for creating widget bindings.

You can add graphic visualizations to your Px page by dragging pre-made widgets from a palette or by making new widgets. When you make a widget using the Make Widget wizard, the wizard dialog box provides fields for creating the widget bindings. The basic procedure is described here. For more details, see the additional procedures for making widgets, as well as [About the Make Widget wizard, page 102](#).

Step 1. From the Nav side bar, drag and drop a widget onto the Px Editor canvas.

The Make Widget wizard appears, with the ORD for the selected component displaying in the **ORD** area. Double-click the ORD value to change it.

Step 2. From the **Source** options list, select the desired type of binding (choose from: Bound Label, Properties, Actions, and others).

---

**NOTE:** Binding options that are dimmed indicate invalid options for the selected component.

---

The **Secondary view** area displays fields and options that are related to the selected Source option.

Step 3. In the **Secondary view** area, choose a widget template, formatting options for display labels, or views, as appropriate for your Source option.

Step 4. In the **Properties** area, edit the properties or actions, as desired.

Step 5. Click **OK** to complete the wizard.

The wizard disappears and your new widget displays on the Px Editor canvas.

Step 6. Toggle the view using the View/Edit Mode toolbar icon to display the widget in the Px Viewer.

## Add WeatherReports on Px Views

You can include one or more iconic displays of weather data on Px views.

### Prerequisites:

- The **weather** module must be installed
- Station has Internet connectivity, or is on a NiagaraNetwork with a Supervisor using the WeatherService

---

**NOTE:** For more information, see the *NiagaraAX Weather Guide*.

---

---

## Add a WeatherReport on a Px page

---

Step 1. In the Palette side bar, open the **weather** palette.

Step 2. Drag the **WeatherService** component to your Services node.

Step 3. Double-click on the **WeatherService** to open the Weather Manager.

Step 4. Click **New**.

The **New** dialog box displays.

Step 5. Click **OK**.

The **New** dialog box closes and a new **WeatherReport** component appears in the WeatherService node. The WeatherReport is configured with the default provider, `NwsWeatherProvider`.

Step 6. Drag the **WeatherReport** component to your Px page.

The **Make Widget** wizard displays.

Step 7. Click **Workbench View** and click **OK**.

Step 8. Click **PxEditor > Toggle View/Edit Mode** to switch to PxViewer

The Px View includes a display of current weather data such as the one shown here:



## Animate a widget property

“Animating” a property means to link a widget property to a bound data component value, so that the widget can display any change in value as it occurs. For more details, see [About Animating Graphics, page 81](#)

---

**NOTE:** Before you can animate a widget property, you must add a binding from the desired component to the target widget. The binding must exist before the property can be animated. For more details, see [Add data bindings, page 6](#)

---

Step 1. In the Px Editor canvas, select (or double-click) the widget that you want to animate.

The widget properties display in the properties side bar (or dialog box, if you double-clicked).

Step 2. In the properties side bar (or dialog box) right-click on the properties field of the property that you want to animate and select **Animate** from the popup menu.

The **Animate** dialog box appears. If you have more than one binding associated with the widget, all bindings will be available in the **Binding** options list.

Step 3. From the **Binding** options list, select the binding that is associated with the component value to which you want to link.

Step 4. From the **Converter Type** options list, select the binding that is associated with the component value you want to link to.

---

**NOTE:** The default converter type is based on the component that you have selected and most of the time this will be the converter type that you want to use. For more details, see [Types of Converters, page 87](#)

---

The **Animate** dialog box displays different fields, based on the type of value (and converter) that you are animating.

Step 5. Complete the binding to the value using the controls in the **Animate** dialog box and click **OK**.

The Animate dialog box disappears and the binding appears in the properties side bar (or dialog box).

Step 6. If using the Properties dialog box, click **OK** to close it.

The property value is animated.

## Animate using static SVG images

This type of animation requires at least two SVG images, of identical size and shape. The animation is caused by alternately displaying one image and then the next. You can apply this animation to a Picture widget or Label.

### Prerequisites:

- Target platform is running NiagaraAX 3.8
- A recent HTML5 compliant browser (issued 2011 or later), for example: FireFox, Chrome, Opera, IE (v.10 or later) and Safari.
- **svgBatik** module (`svgBatik.jar`) must be in the Modules folder.
- SVG images must be located in the station file system

The following example uses the Picture widget and two arrow images, in one the arrow points up, in the other it points down. For more information, see [About SVG support, page 62](#).

---

**NOTE:** Prior to performing this procedure, copy any SVG images you intend to use to the station file system.

---

### Animate a Picture widget using two static SVG images

---

Step 1. In the station, create a new **EnumWritable** to use as the data source for the animation.

Step 2. Set the range values for the **Facets** as follows:

- 0 = Up
- 1 = Down

Step 3. Right-click the component and select **Views > New View** to open a new Px view.

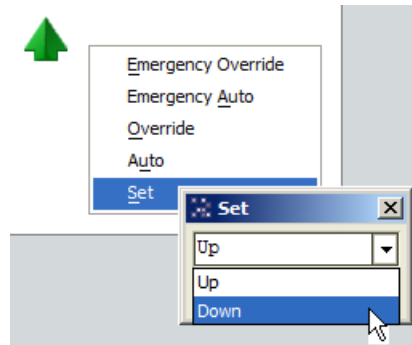
Step 4. Click **PxViewer > Toggle View/Edit mode** to switch to Edit mode.

Step 5. In the Palette side bar, open the **bajau** palette, expand the Widgets folder and drag the **Picture** widget to the Px Editor canvas

Step 6. Double-click the widget to open the **Edit Properties** dialog and click the **Add Binding** button (in the upper right corner), to add a value binding to the widget.

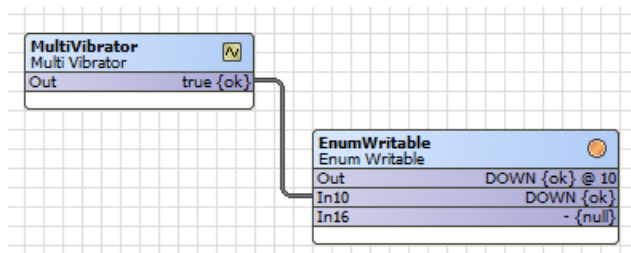
Step 7. In the Value Binding section of the **Properties** dialog box, click on the ORD property of the binding and click the selection icon () to access the Component Chooser.

- Step 8. In the Component Chooser, locate and select the “out” value of the **EnumWritable** data source created in step one, for example: station:|slot:/Drivers/Points/EnumWritable/out/value, and click **OK**
- Step 9. In the Picture widget section of the **Properties** dialog, right-click the **image** property and click **Animate**.
- Step 10. In the **Animate** dialog, select the first facet of the **EnumWritable** and click **Set** and use the File Chooser to locate and select the first image file (for example, !stations/PxTest2/Files/px/arrowUp.svg) to be displayed for facet value “0” and click **OK**.
- Step 11. Select the second facet of the **EnumWritable** and click **Set** and use the File Chooser to locate and select the second image file (for example, !stations/PxTest2/Files/px/arrowDown.svg) to be displayed for facet value “1” and click **OK**
- Step 12. Click **Save** to save these changes to your Px View.
- Step 13. Switch to View mode (click **PxEditor > Toggle View/Edit mode**).
- Step 14. In the canvas pane, right-click the displayed arrow image, select **Actions > Set** and change to **Down** as shown here:



The graphic image changes automatically when you set the **Actions**. The image displayed depends on whether the **EnumWritable** Action is set to Up or Down.

Additionally, providing an input to the EnumWritable using a control component, such as **MultiVibrator** in the **kitControl** palette, causes the animation to occur automatically. For example, drag and drop a MultiVibrator component onto the wire sheet view for the EnumWritable and set the MultiVibrator **Period** property to 5 seconds (so that the **out** value changes from true to false every five seconds). In the wire sheet view, connect the MultiVibrator “out” value to the EnumWritable “In” as shown here:



Switch to the Px Editor View mode to observe the animation.

## Create a “global” navigation menu using PxInclude

Create a “global” navigation menu that you can use on multiple Px pages in a station to achieve a standardized, uniform look.

You can have a consistent navigation menu available on your home page or on all pages and you need only edit the navigation menu page any time a change is required. Since the navigation

menu Px file is a PxInclude on the other Px views, they are automatically updated with any change that you make.

### Create a navigation menu for use on multiple Px pages

Step 1. Create a new Px view for the menu.

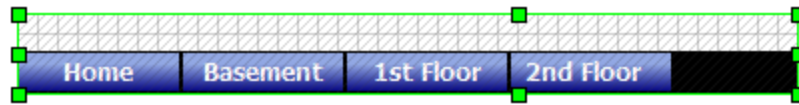
**NOTE:** Remove the ScrollPane at the root in this view otherwise scroll bars will display on the menu once it is embedded in other Px views.

Step 2. Drag and drop **Action (Image) button** widgets from the **kitPx** palette to the canvas pane, one for each Px view that will include the menu.

Step 3. For each image button widget, add a value binding setting the **ord** property to the appropriate Px file. For example, `Home.px`, `Basement.px`, `1st_Floor.px`, `2nd_Floor.px`.

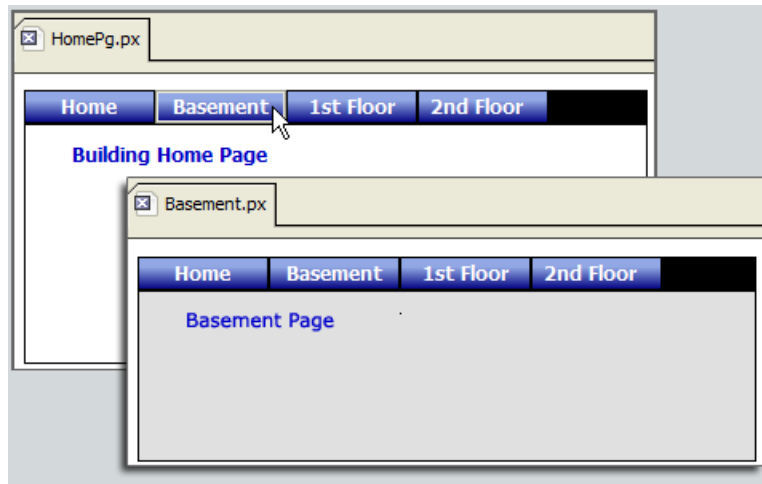
Step 4. Click the **Save** icon in the Workbench toolbar to save your changes.

This Px file is the global, navigation menu that you can include in other Px views.



Step 5. To embed the navigation menu, drag and drop the **PxInclude** widget from the **ba-jai** palette to each of the other Px pages on the station.

The result is a standardized menu displayed on each page, as shown below, which you maintain simply by editing the navigation menu page any time a change is required.



## Create a scalable image using the Picture widget

In AX-3.8, you can easily create a scalable image using an SVG image in a Picture widget with the scale property set to Fit.

### Prerequisites:

- Target platform is running NiagaraAX-3.8
- A recent HTML5 compliant browser (issued 2011 or later), for example: FireFox, Chrome, Opera, IE (v.10 or later) and Safari.
- **svgBatik** module (`svgBatik.jar`) must be in the `!modules` folder.

- SVG images must be created outside of Workbench, using vector graphic software, and copied to the station file system

SVG images are "vector graphic" images. You can certainly save images in other graphic formats, such as BMP, GIF, PNG, etc., as SVG images. However, it is important to realize that if those images are not a vector graphics format to begin with, the image quality will degrade when scaled.

---

**NOTE:** Interactive features of SVG images, such as embedded Javascript-based animations, are not supported.

---

- Step 1. In the Nav tree, right-click a component and select **Views > New View** to create a new Px view.
- Step 2. In the Palette sidebar, open the **bajauri** palette and expand the **Widgets** folder.
- Step 3. Drag the **Picture** widget to the Px Editor canvas.
- Step 4. Double-click the **Picture** widget to display the **Properties** dialog box.
- Step 5. Click the File Chooser icon to the right of the `image` property field and browse to select your SVG image and click **OK**.
- Step 6. Set the **scale** property to `Fit` and click **OK**.
- Step 7. In the Px view, click and drag a corner of the widget to resize it smaller and/or larger. The Picture widget scales perfectly without degrading the graphic quality.

## Embed a history chart in a Px view

Embedding a pre-configured, individual history chart in a Px view.


### Prerequisites:

- A pre-configured history in the station's History node.

### Embed a pre-configured history chart in a Px page

- Step 1. Drag the pre-configured history or history extension component onto your Px page.
- Step 2. In the Make Widget wizard, select **Workbench view** and `History Table`.
- Step 3. Click **OK** and click **Save** in the Workbench menu bar.

The individual history is embedded in the Px page.

You can use the export icon () or the Workbench menu **File > Export** command to export a history chart or history table. The history name and time range of the data are included at the top of the exported file (CSV, TXT, PDF), however, it will show "value" as the column identifier.

If you also want to provide access to the History Chart Builder in your Px view, you can add a popup binding on a widget that launches it. For details, see [Launch the History Chart Builder in a Px view](#), page 16.

## Embed Px files with ORD variables using PxInclude

This example illustrates embedding Px files with ORD variables using a PxInclude widget.

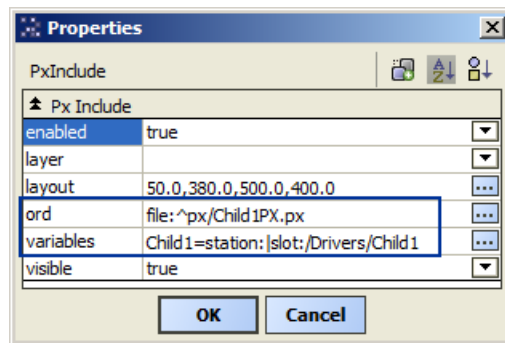
### Prerequisites:

- Have two relativized Px views

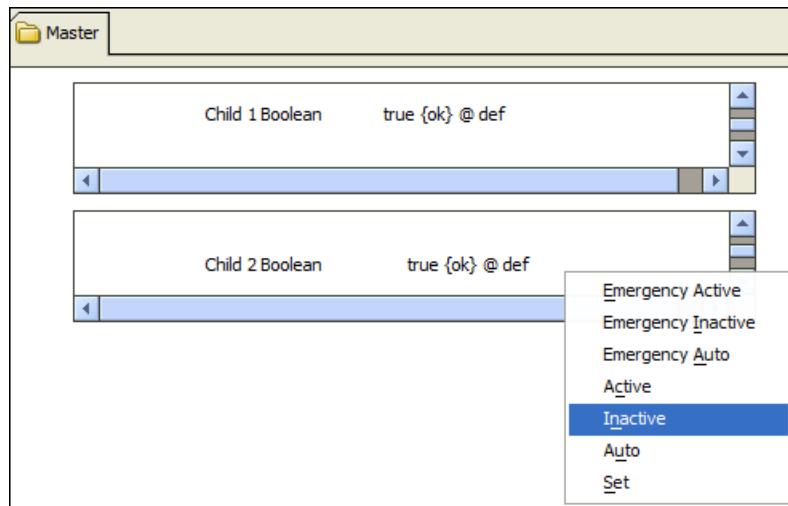
In this example, there are two relativized views, Child 1 and Child 2, that will be included on a "master" Px file.

## Embed Px files with ORD variables in a “master” Px file using PxInclude

- Step 1. Edit the relativized ORDs in your Px views to use the `$(variablename)` syntax in place of the actual ORDs, for example: `$(Child1)/Variable1`.
- Step 2. Save the Child 1 and Child 2 Px files.
- Step 3. Create another folder named, Master and add a Px view.
- Step 4. Drag the Child 1 folder onto the Master Px view.
- Step 5. In the **Make Widget** wizard, select the **Include Px File** option, and use the File Ord Chooser to select the “Child1Px.px” file and click **OK**.
- Step 6. Select variable to bind, `Child1`, and click **OK**. The PxInclude widget’s properties are shown below.



- Step 7. Repeat steps 4–8 above using the Child 2 folder.
- Step 8. Switch to View mode to see the embedded Px files with Ord variables. Use the **Actions** menu to affect changes, as shown here:



## Group objects using Px Layers

Use Px Layers to group objects in a Px Editor view for ease of manipulation.

When you assign several objects to a common layer, you can easily select all of the objects simply by selecting the layer. You can also lock and/or hide all of the objects in a layer. For more details, see [About Px Layers, page 61](#)

Typically you work with Px Layers in the following ways:



### **Add a Px Layer**

Add a new Px Layer in Px Editor view.

#### **Prerequisites:**

- You must have the **Px Layers** palette before you can assign any objects to a layer.

This procedure describes how to create a new Px Layer.

---

**NOTE:** Objects do not display a layer property until at least one layer exists.

---

Step 1. In Px Editor, on the **PxLayers** palette, click the **Add** () button.

Step 2. In the **Add** dialog box, type a name for the new layer and Click the **OK** button

The new layer displays in the Px Layers palette and all editable objects have a new “layers” property value option available.

### **Assign/Unassign objects to a layer**

Add, change or remove an object’s layer assignment in the Px Editor view.


This procedure describes how to add, change or remove an object’s layer assignment.

Step 1. Right-click on the desired object on the Px Editor canvas or in the Widget Tree palette) and select **Edit Properties** from the popup menu.

---

**NOTE:** Instead of using the popup menu, you can select an object and change the layers value using the Properties palette.

---

Step 2. In the Properties dialog box, click the **Value Select** button ()

Step 3. From the option list, choose one of the following:

- a named layer value option to add or change the object’s layer assignment
- the empty (top) option to remove any object-layer assignment

Step 4. In the **Properties** dialog box, click **OK**.

The dialog box closes and the object-layer assignment is added, changed or removed, depending on your selection.

### **Rename a Px Layer**

This procedure describes how to change the name of a Px Layer.

---

**NOTE:** Renaming a layer does not affect the layer assignments of the objects assigned to that layer. For example, if Object A is assigned to Layer 1 when Layer 1 is renamed to Layer 2, Object A remains assigned to the layer (now named Layer 2).

---

Step 1. In the Px Editor’s **Px Layers** palette, right-click on the layer that you want to rename and select the **Rename** command from the popup menu.

Step 2. In the **Rename** dialog box, type the new name and click **OK**.

The layer name is changed.

### **Delete a Px Layer**

Remove a Px Layer in the Px Editor view.

This procedure describes how to delete a Px Layer.

---

**NOTE:** Removing a layer deletes the layer from the Px Layers palette but does not delete any objects or affect any object properties other than the “layer” property for any assigned objects. The layer property value is removed from any objects assigned to the deleted layer.

---

Step 1. In the Px Editor’s **Px Layer** palette, right-click on the layer that you want to delete and select **Remove** from the popup menu.

The layer is deleted and no longer appears in the Px Layers palette.

## Launch the History Chart Builder in a Px view

You can launch the History Chart Builder in a Px view by adding a Pop Up binding on a widget in your Px page. The ORD of the Popup binding is linked to the station History. When finished with the chart builder, simply close the Popup to return to your Px view.

### Prerequisites:

- A pre-configured history in the station’s **History** node.

### Launch the History Chart Builder in a Px view using a Pop Up binding on a widget

Step 1. Drag and drop an **Image button** (Action button) widget from the **kitPx** palette to your Px page.

Step 2. In the **Image button** property sheet view, configure the following:

- in the **text** slot, type the desired button text (for example, **Popup Chart Builder**)

Step 3. Add a Popup Binding

- in the **ord** slot, paste the following value: `history: |view:history: HistoryChartBuilder`

Step 4. Click **OK** and click the **Save** icon in the Workbench menu bar.

Step 5. In View mode, click the **Popup Chart Builder** button to access the **History Chart Builder** in the Popup window.

---

**NOTE:** You can control the size, position and title of the Popup window.

---

Step 6. When finished looking at the histories, close the Popup window, and the primary Px view is still open.

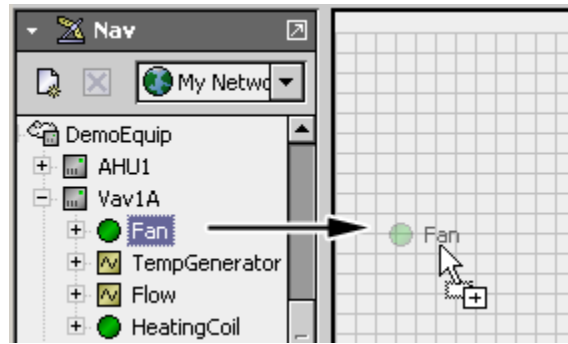
## Make Bound Label widgets

This example illustrates how to create a Bound Label widget using the Make Widget wizard.

### Make a Bound Label widget

Step 1. In the Nav side bar, expand the tree to locate the component that you want to add to the Px Editor canvas.

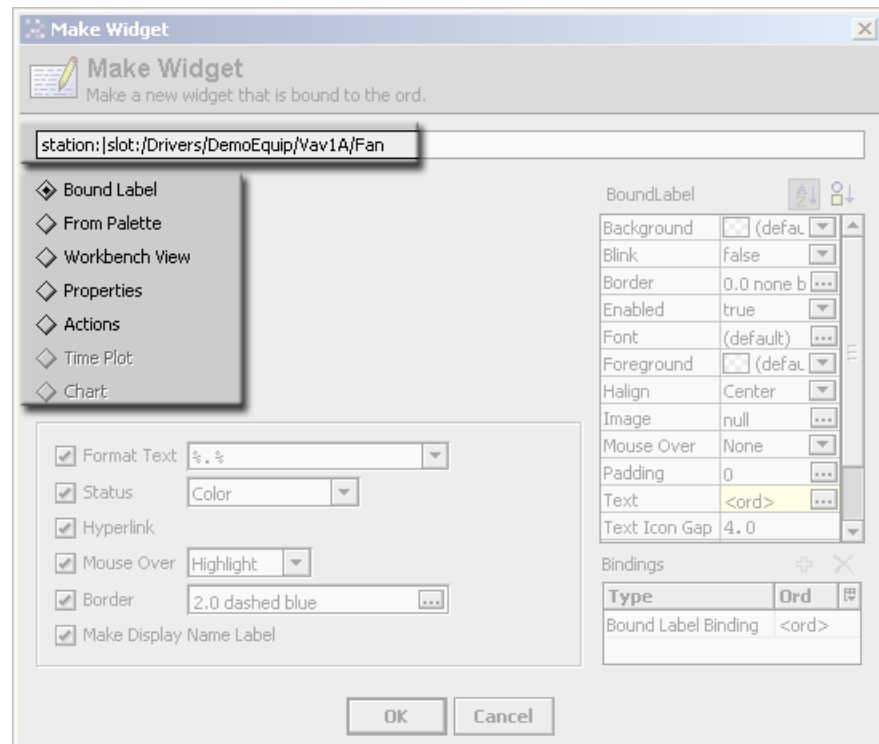
Step 2. Drag and drop the desired component onto the canvas, as shown here.



The Make Widget wizard displays, as shown.

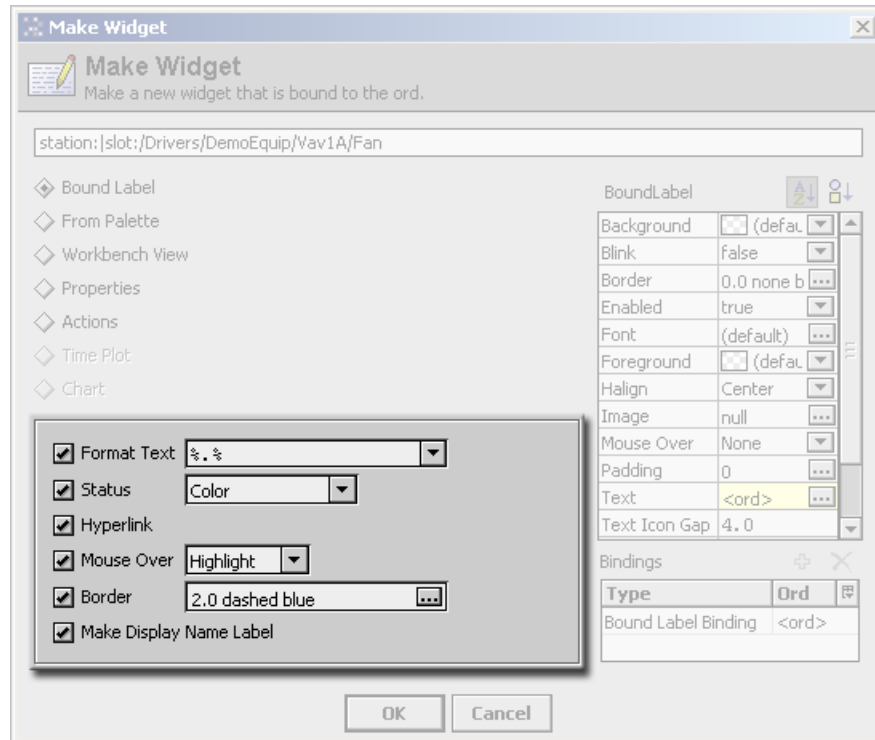
**NOTE:** The ORD for the selected component displays in the ORD area. You can double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component.

Step 3. Select the Bound Label option in the **Source** options list as shown here.



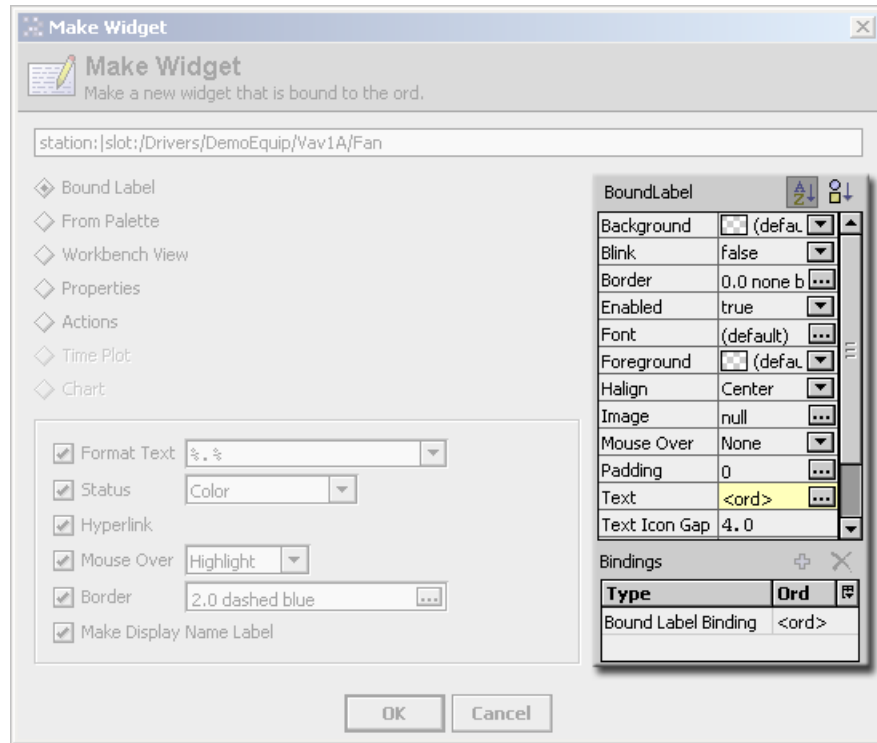
The **Secondary view** area displays a list of property options that are available.

Step 4. In the **Secondary view** area, select the options that you want and set their properties using the adjacent property options lists, as shown below.



**NOTE:** The **Make Display Label** option creates a separate unbound label that appears directly beside the bound label when the widget displays on the canvas. Edit the display label properties using the properties side bar after completing the **Make Widget** wizard.

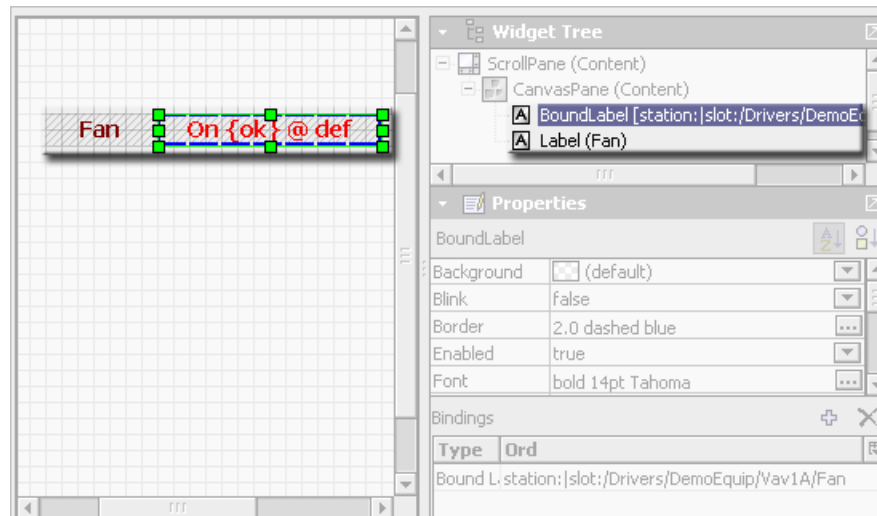
Step 5. In the **Properties** sheet area, shown below, edit the bound label properties.



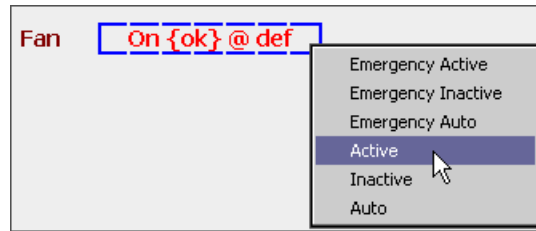
**NOTE:** Some of the options in the Secondary view area are also editable in the **Properties** sheet area. It is possible to override an option by changing it in the properties sheet area.

Step 6. Click the **OK** button to complete the wizard.

The Make Widget wizard disappears and the new bound label appears on the Px Editor canvas as shown here.



Step 7. Toggle the view using the **View/Edit Mode** toolbar icon to display the widget in the Px Viewer, as shown below.

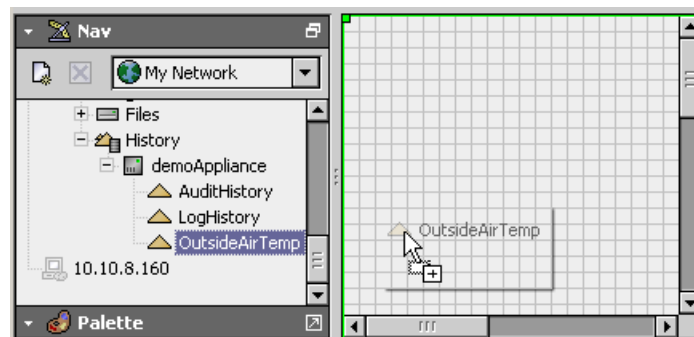


## Make Chart widgets

This example illustrates how to create a chart widget using the Make Widget wizard.

### Make a Chart widget

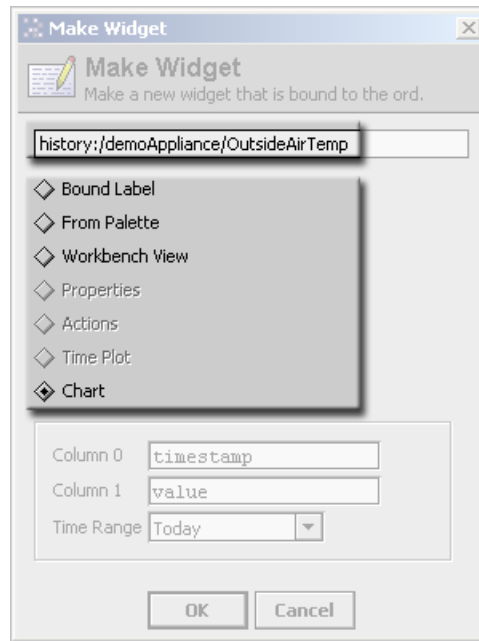
- Step 1. Open the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the desired component onto the canvas, as shown here.



The Make Widget wizard displays, as shown.

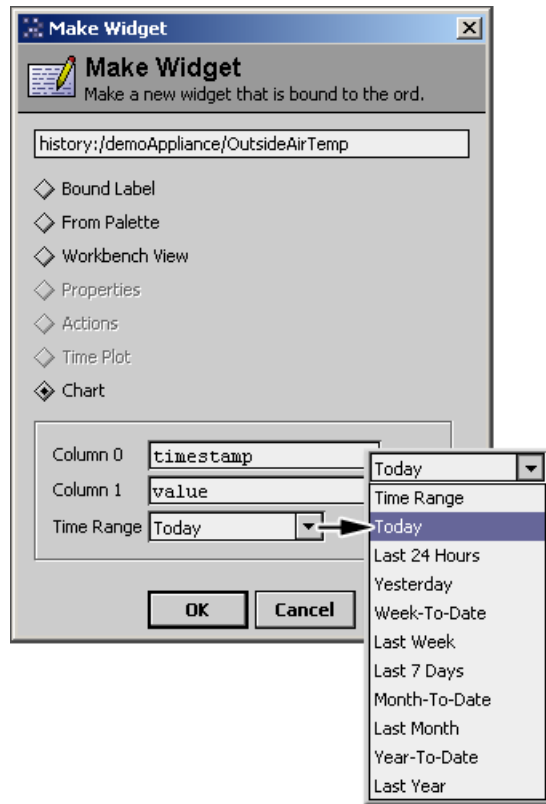
**NOTE:** The ORD for the selected component displays in the ORD area. Double click the ORD to browse for a different slot in the component (for example: “out” “value” or to bind to a different component, if desired).

- Step 3. Select the Chart option in the **Source** options list as shown here.

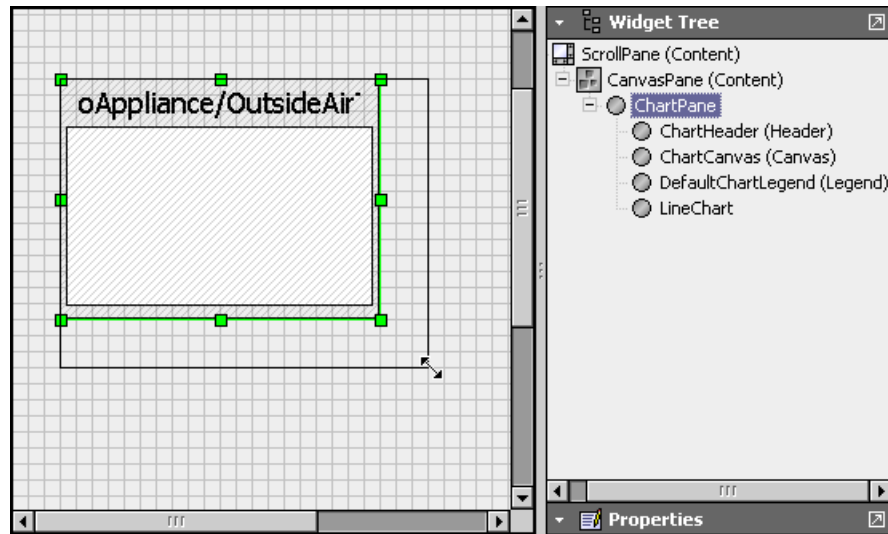


The **Secondaryview** area displays fields that allow you to choose the data for the x and y axes of the chart and the Time Range.

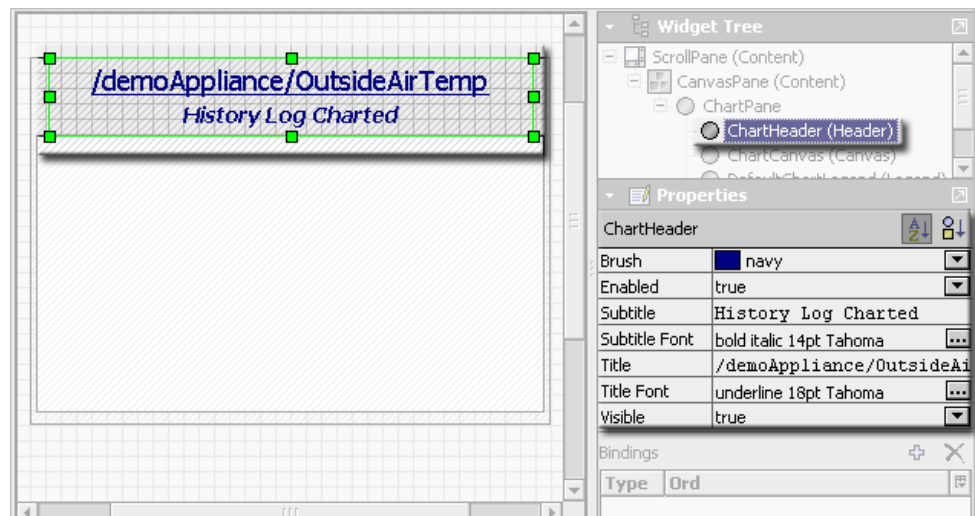
- Step 4. In the **Secondaryview** area enter the value source for column 0 and for column 1. Also, choose the desired range from the time range option list, as shown.



- Step 5. Click the **OK** button to complete the wizard. The Make Widget wizard disappears and the new Chart widget appears on the Px Editor canvas as shown here.

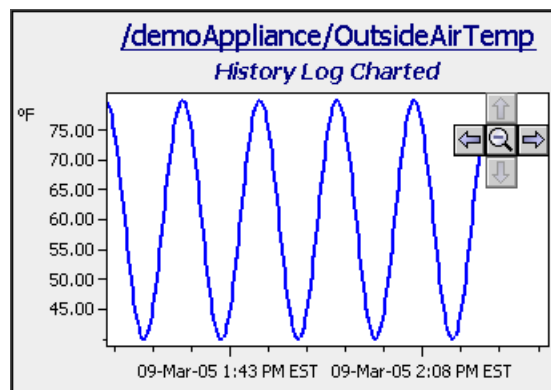


Step 6. Edit the widget, if desired, by doing one or both of the following:



- Resize the widget, as desired to allow the plot to display completely.
- Edit the Chart widget properties (header, lines, fonts, and so on) using the Widget Tree and Properties palettes, as shown in the above figure.

Step 7. Toggle the view using the View/Edit Mode toolbar icon to display the Chart in the Px Viewer, as shown.



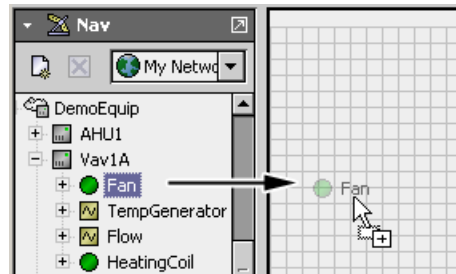


## Make component Properties-based widgets

This example illustrates how to create a component Properties-based widget using the Make Widget wizard.

### Make a component Properties-based widget

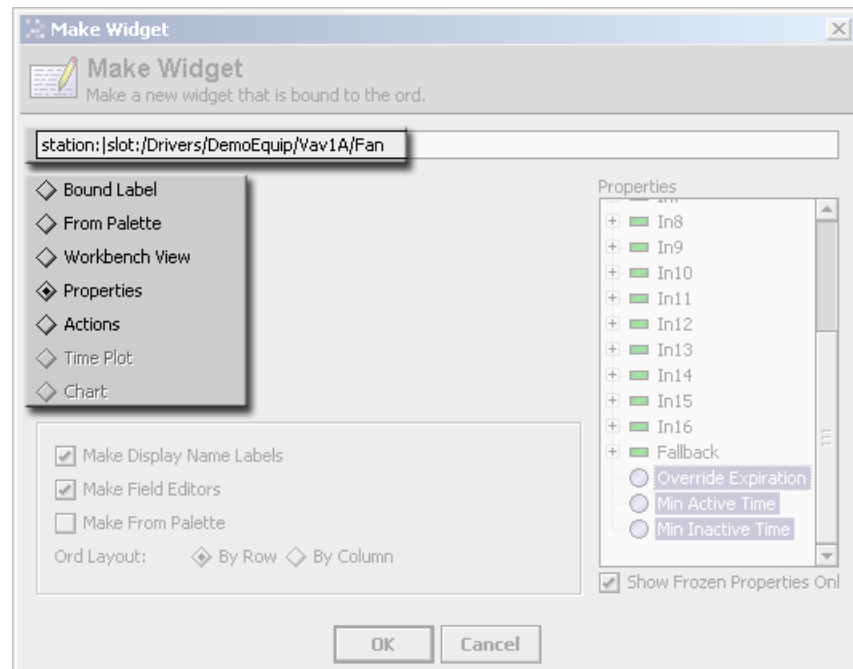
- Step 1. In the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the desired component onto the canvas, as shown in below.



The Make Widget wizard displays, as shown.

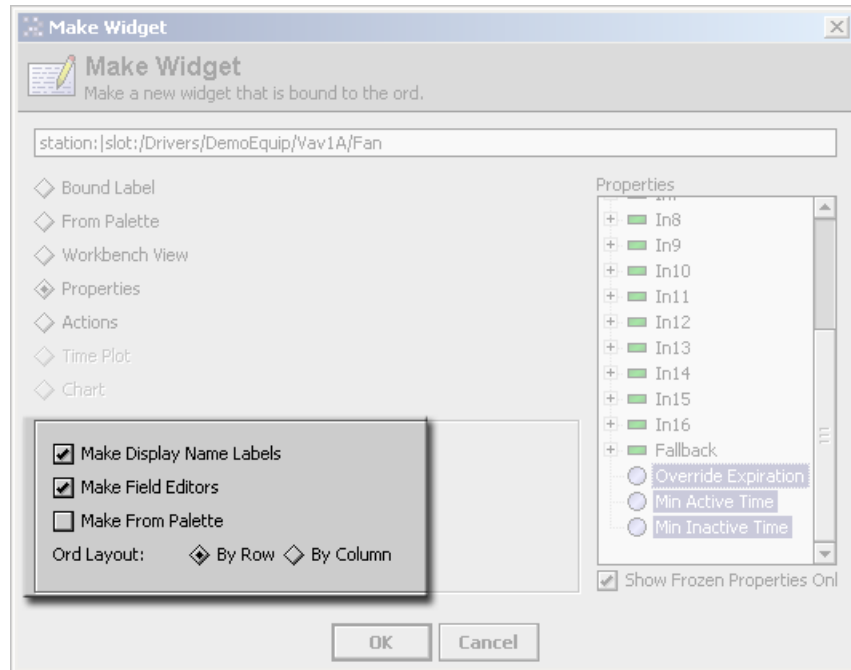
**NOTE:** The ORD for the selected component displays in the ORD area. Double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component, if desired.

- Step 3. Select the **Properties** option in the **Source** options list as shown here:



The **Secondary view** area displays options that are available for creating the new widget.

- Step 4. In the **Secondary view** area, select one or more options from the list, as shown here:



- **Make Display Name Labels**

Choosing this option displays a separate unbound label that appears directly beside the bound label when the widget displays on the canvas. Edit the display label properties using the properties side bar after completing the Make Widget wizard

- **Make Field Editors**

Choosing this option displays an area that allows you to edit the property in the Px viewer (or in a browser).

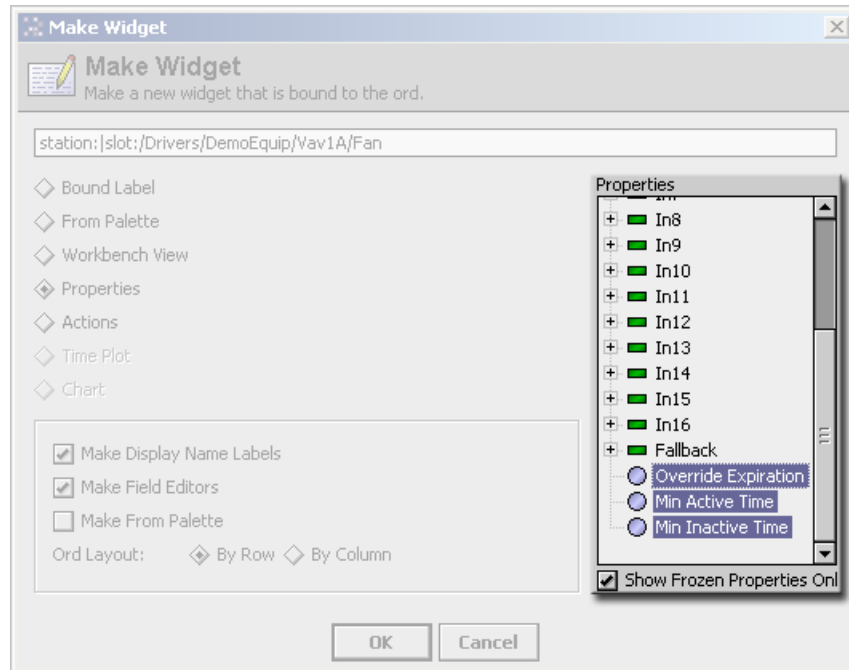
- **Make from palette**

This option displays the palette side bar view in the wizard and allows you to find and use a suitable widget template from any available palette.

- **Ord and Property Layout options (By Row or By Column)**

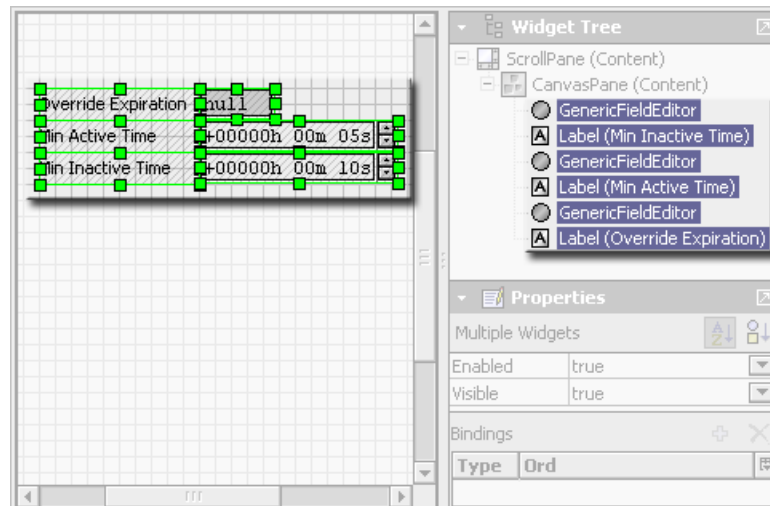
These options affect how the fields will be arranged on the canvas when you complete the wizard. Your choice of layout is optional and you may rearrange the layout after you place the items on the canvas.

Step 5. In the **Properties** sheet area, select the properties that you want to bind to the widget. Hold down the **Ctrl** key to select more than one property, as shown here:

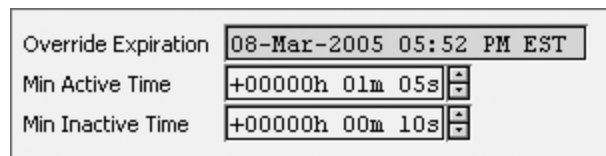


Step 6. Click the **OK** button to complete the wizard.

The Make Widget wizard disappears and the new property widgets appear on the Px Editor canvas as shown.



Step 7. Toggle the view using the View/Edit Mode toolbar icon to display the widgets in the Px Viewer, as shown here:

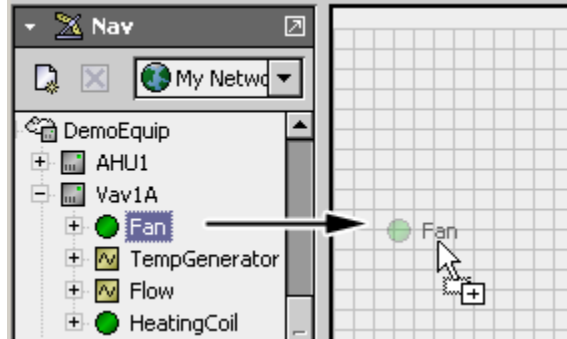


## Make Palette kit-based widgets

This example illustrates how to create a Palette kit-based widget using the Make Widget wizard.

**Make a Palette kit-based widget**

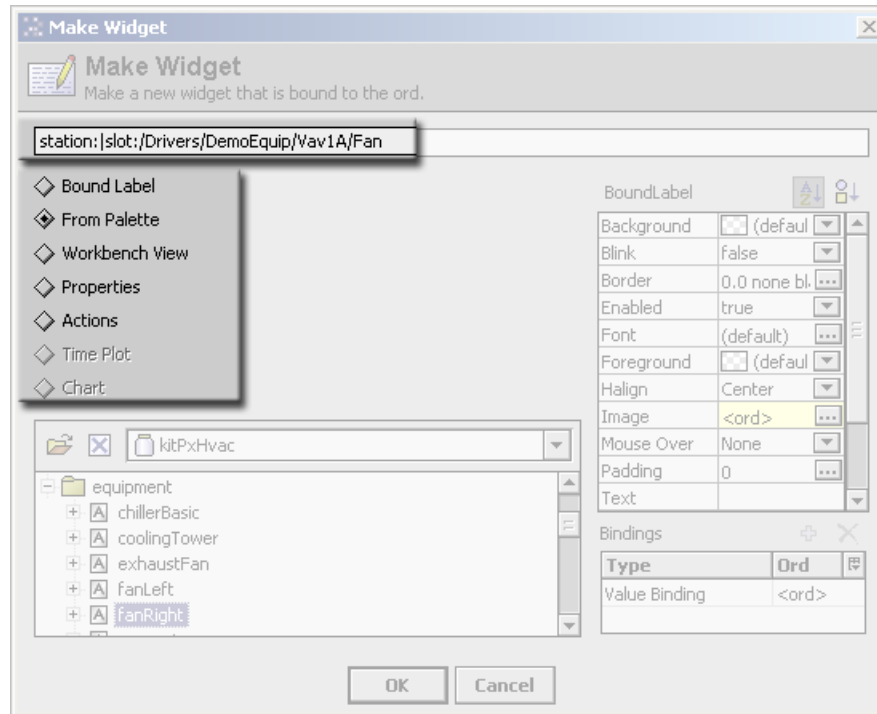
- Step 1. In the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the component onto the canvas, as shown here.



The Make Widget wizard displays.

**NOTE:** The ORD for the selected component displays in the ORD area. You can double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component.

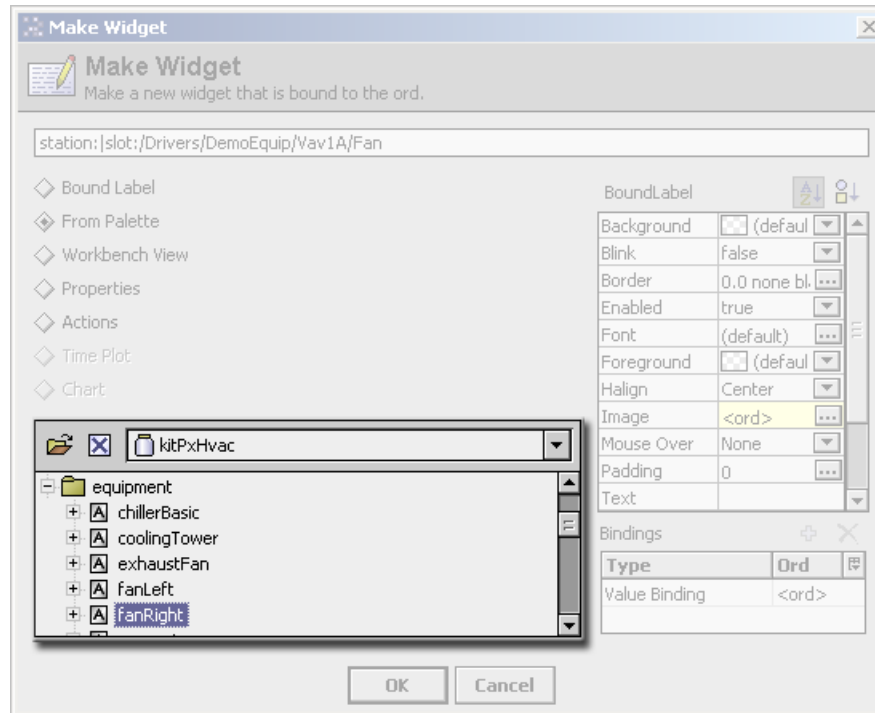
- Step 3. Select the From Palette option in the **Source** options list as shown below.



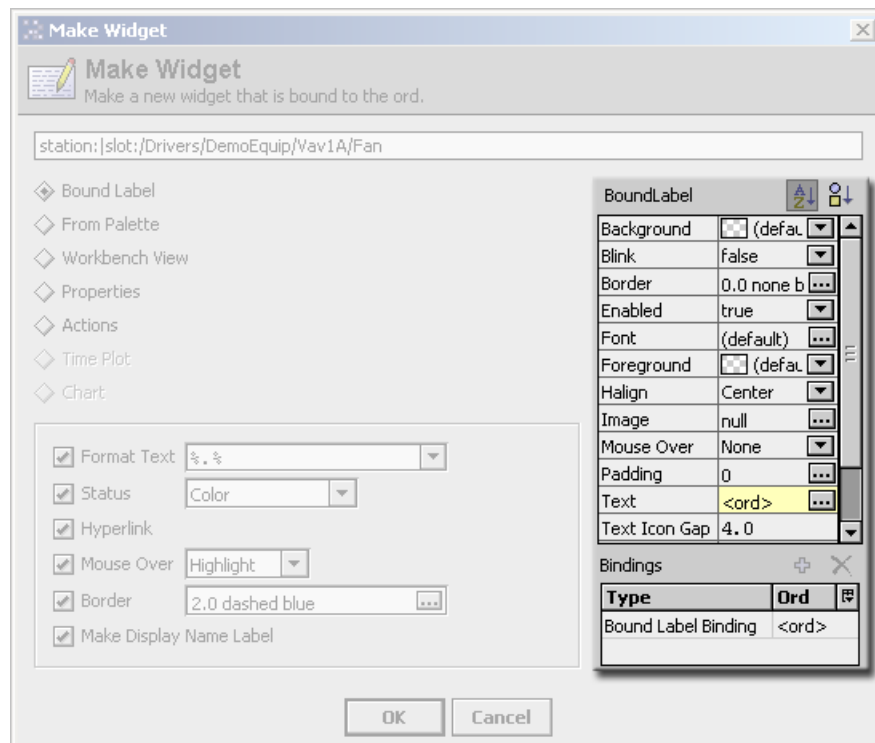
The **Secondaryview** area displays the palette side bar view.

**NOTE:** All palettes that are currently open in the palette side bar are available under the module option selector arrow. Other palettes may be opened by clicking on the folder icon.

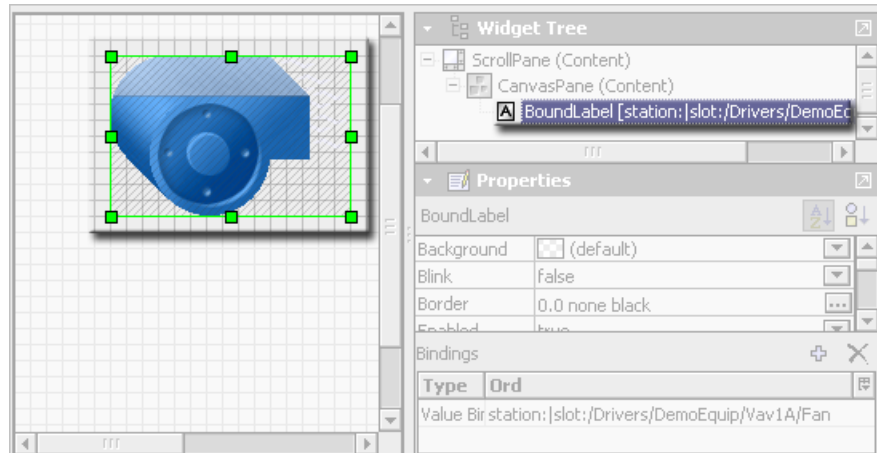
- Step 4. In the **Secondary view** area, expand the module tree to find and select the desired widget template, as shown here.



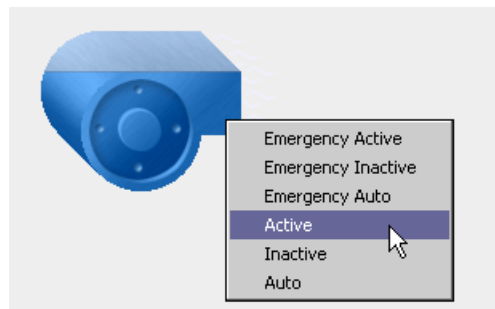
- Step 5. In the **Properties** sheet area, shown here, edit the widget template properties.



- Step 6. Click the **OK** button to complete the wizard. The Make Widget wizard disappears and the new widget appears on the Px Editor canvas as shown below.



Step 7. Toggle the view using the **View/Edit Mode** toolbar icon to display the widget in the Px Viewer, as shown here.

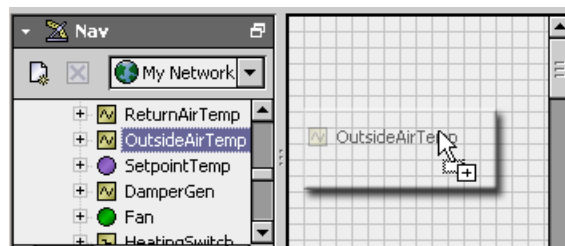


## Make Time Plot widgets

This example illustrates how to create a Time Plot widget using the Make Widget wizard.

### Make a Time Plot widget

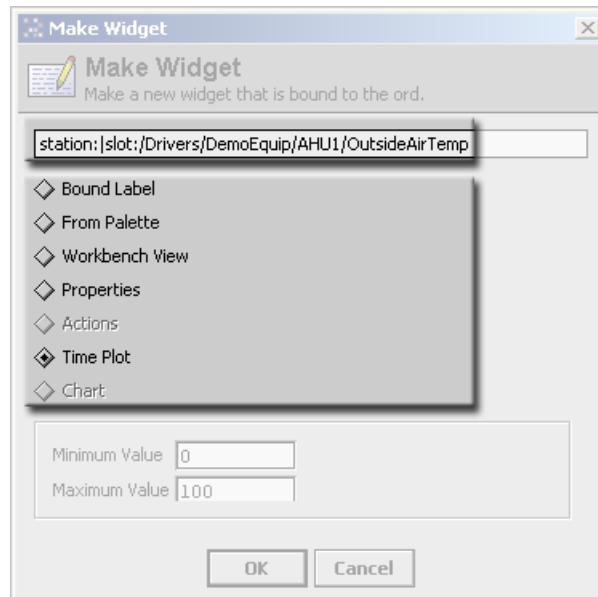
- Step 1. Open the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the desired component onto the canvas, as shown in .



The Make Widget wizard displays, as shown.

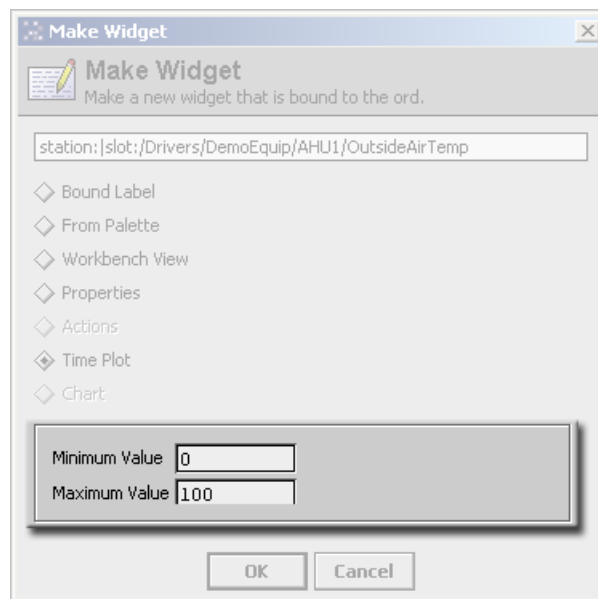
**NOTE:** The ORD for the selected component displays in the ORD area. Double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component, if desired.

- Step 3. Select the Time Plot option in the **Source** options list as shown here.

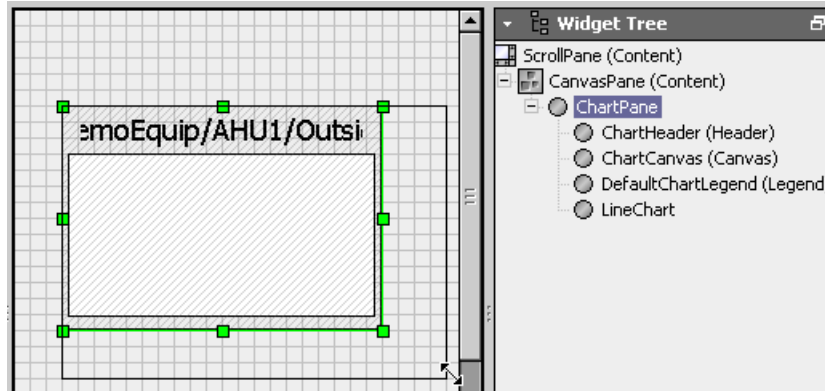


The **Secondary** view area displays Minimum Value and Maximum Value fields.

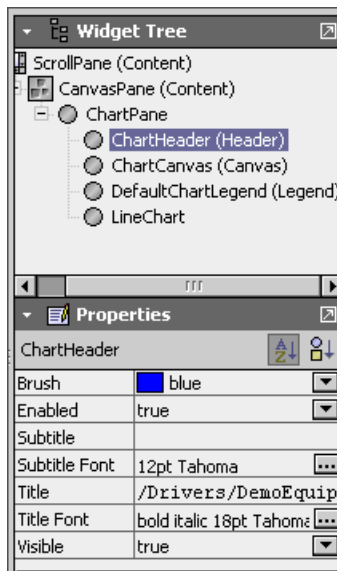
- Step 4. In the **Secondary** view area enter the desired minimum and maximum values that you want to display in the Time Plot widget, as shown.



- Step 5. Click the **OK** button to complete the wizard. The Make Widget wizard disappears and the new Time Plot widget appears on the Px Editor canvas as shown.

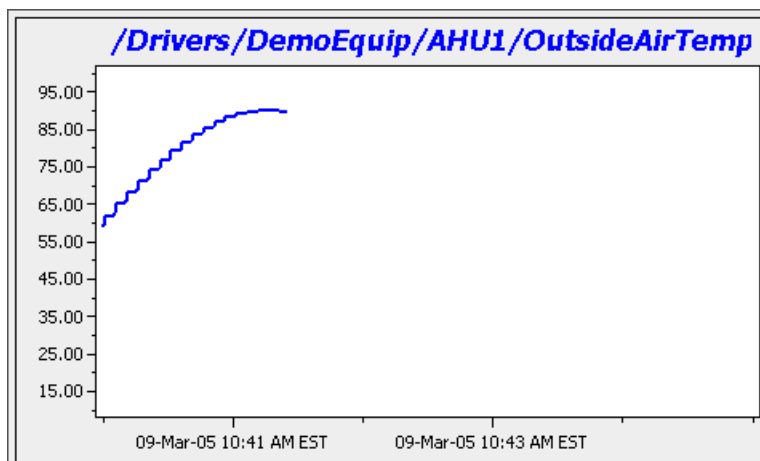


Step 6. Edit the widget, if desired, by doing one or both of the following:



- Resize the widget, as desired to allow the plot to display completely.
- Edit the Time Plot widget properties (header, lines, fonts, and so on) using the Widget Tree and Properties palettes, as shown.

Step 7. Toggle the view using the View/Edit Mode toolbar icon to display the widgets in the Px Viewer, as shown.



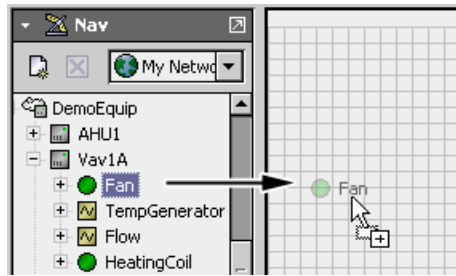


## Make Workbench view-based widgets

This example illustrates how to create a Workbench view-based widget using the Make Widget wizard.

### Make a Workbench view-based widget

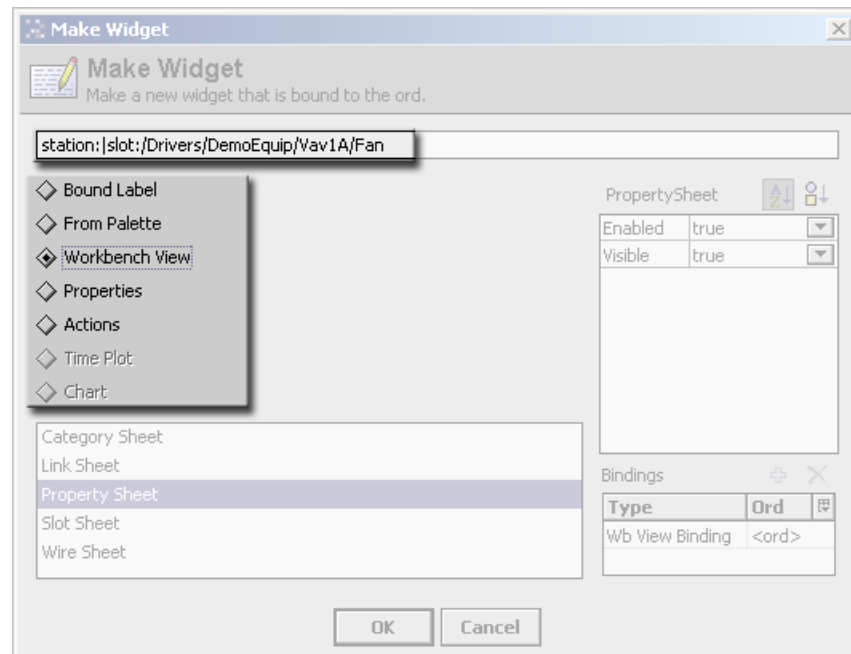
- Step 1. In the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the component onto the canvas, as shown here:



The Make Widget wizard displays, as shown.

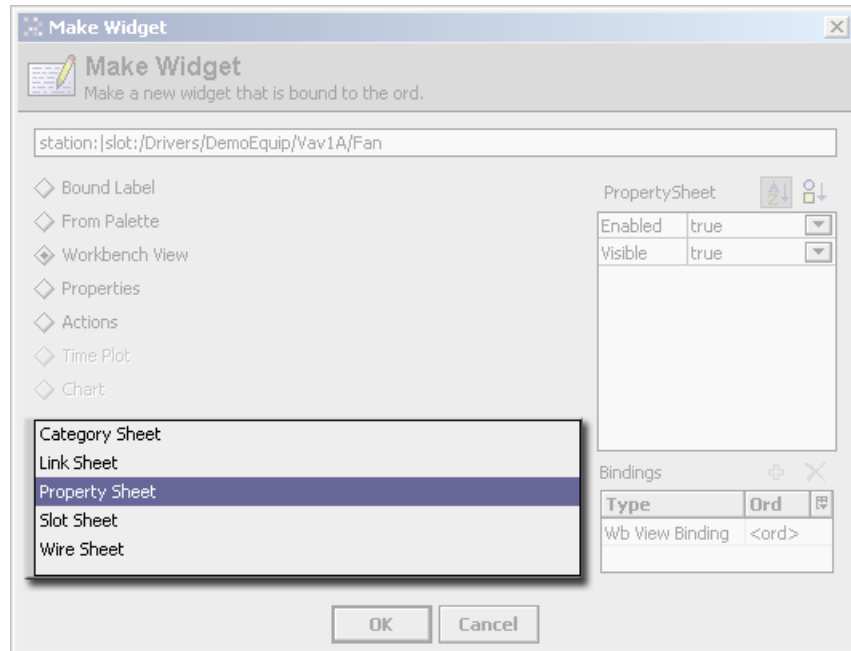
**NOTE:** The ORD for the selected component displays in the ORD area. Double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component, if desired.

- Step 3. Select the Workbench View option in the **Source** options list as shown here:



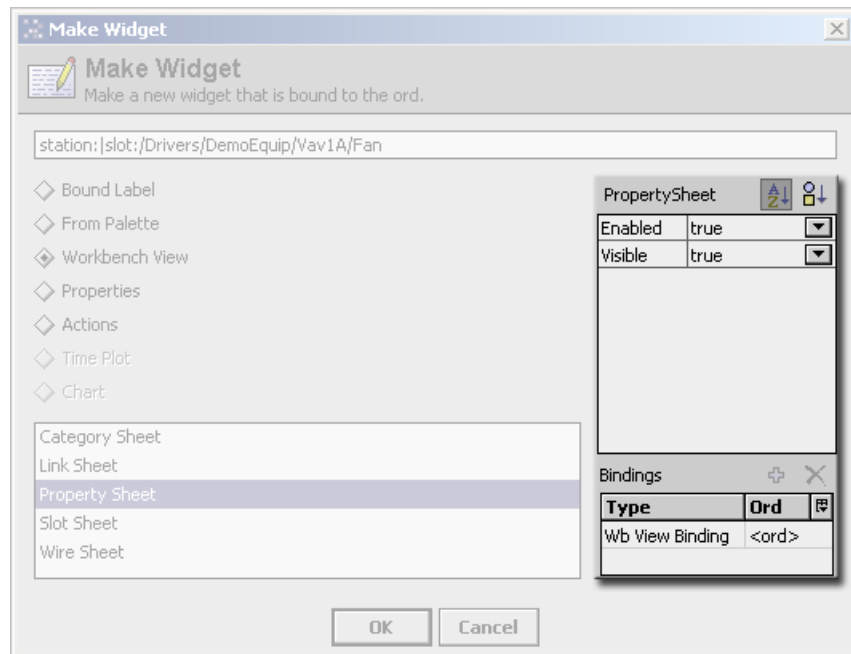
The **Secondary view** area displays a list of view options that are available for the selected component.

- Step 4. In the **Secondary view** area, select the desired component view to add, as shown:



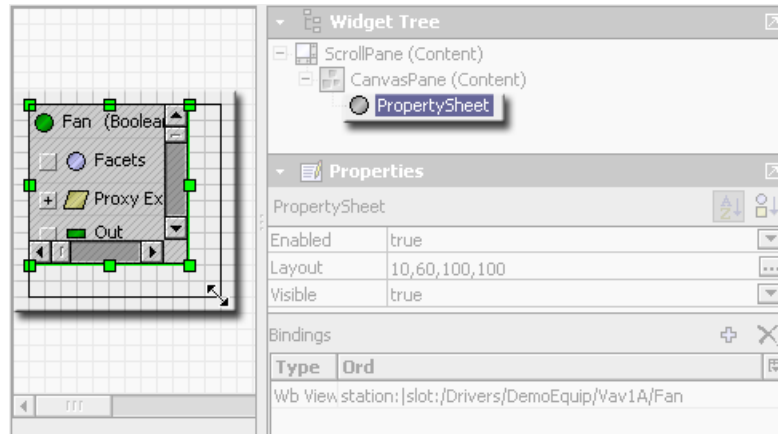
The **Properties** sheet area displays the component properties.

Step 5. In the **Properties** sheet area, shown below, edit the widget template properties, as desired.



Step 6. Click the **OK** button to complete the wizard.

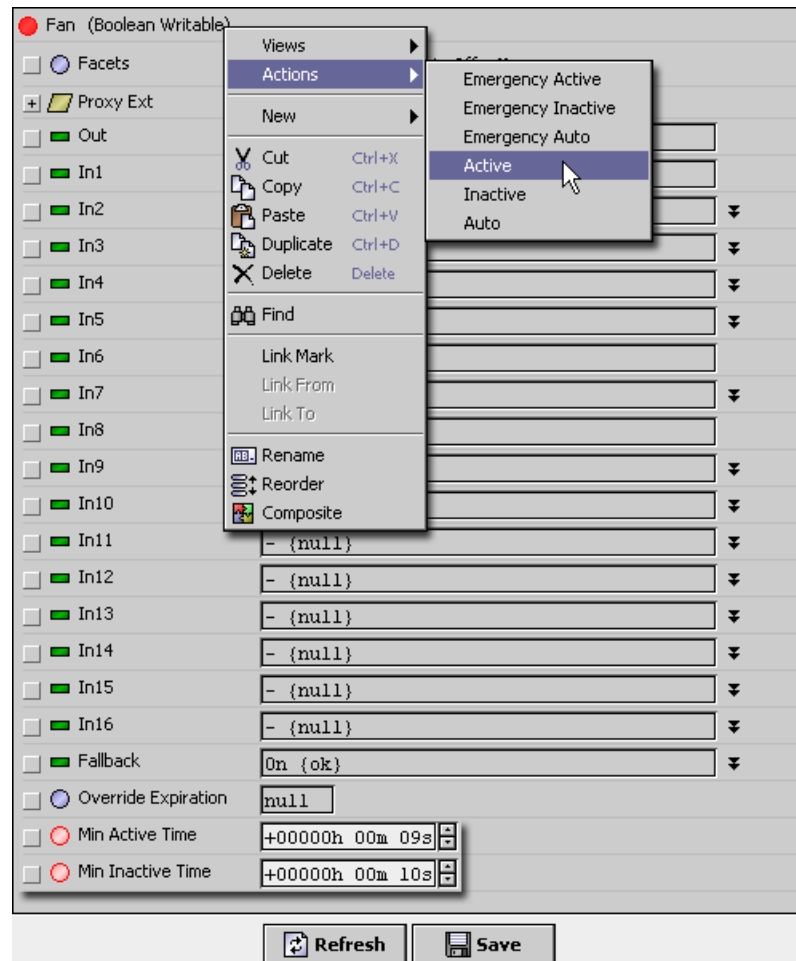
The Make Widget wizard disappears and the new widget view appears on the Px Editor canvas as shown below.



Step 7. Use the handles on the widget view to expand the view to the desired size or edit the Layout field in the properties side bar.

**NOTE:** Scroll bars will appear if the widget view is larger that the canvas view area.

Step 8. Toggle the view using the **View/Edit Mode** toolbar icon to display the widget in the Px Viewer, as shown.



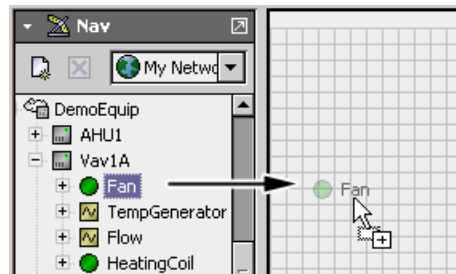
The view is fully functional in the Px viewer, where you can edit properties, invoke commands, or make other edits to a component, just as if you are editing in the Workbench view.

## Make Action widgets

This example illustrates how to create a Action widget using the Make Widget wizard.

### Make an Action widget

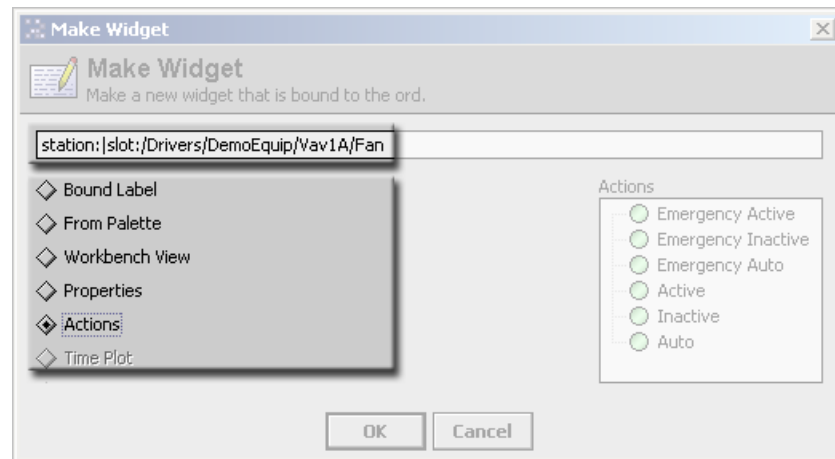
- Step 1. Open the Nav side bar and expand the tree to locate the component that you want to add to the Px Editor canvas.
- Step 2. Drag and drop the desired component onto the canvas, as shown here:



The Make Widget wizard displays, as shown.

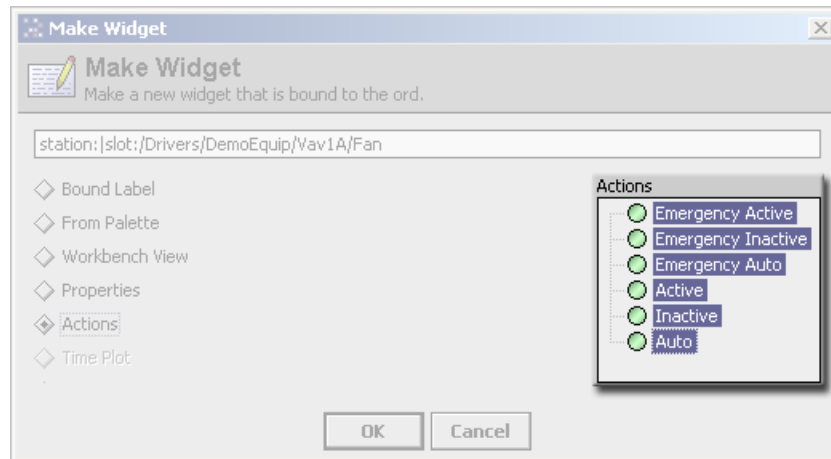
**NOTE:** The ORD for the selected component displays in the ORD area. Double click the ORD to browse for a different slot in the component (for example: “out” or “value”) or to bind to a different component, if desired.

- Step 3. Select the Action option in the **Source** options list as shown below.



The **Properties view** area displays actions that are available for creating the new widget.

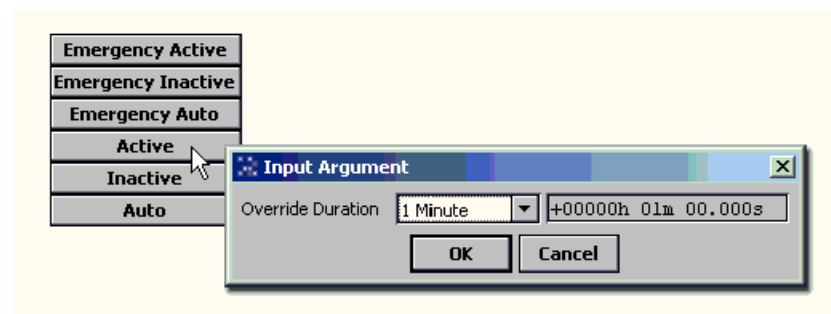
- Step 4. In the **Secondary view** area, select one or more actions from the list, as shown here:



Step 5. Click the **OK** button to complete the wizard. The Make Widget wizard disappears and the new property widgets appear on the Px Editor canvas as shown here:



Step 6. Toggle the view using the View/Edit Mode toolbar icon to display the widgets in the Px Viewer, as shown.



## Relativize absolute ORDs

Open the Relativize ORDs dialog box from the Bound ORDs palette. This Px Editor tool provides a convenient way to change absolute ORDs to relative ORDs.

When you use the Px Editor tools to bind data, the ORD is usually supplied in an absolute format, by default. For example, `station:|slot:/Logic/HousingUnit/AirHandler/DamperPosition`. If you change the data binding to make it relative, then the path will resolve relative to its current parent ORD. This relative path makes the Px file resolve data bindings correctly to identically named components that reside in different locations, making this one Px file reusable in many views. For more information, see [About relative and absolute bindings](#), page 91.

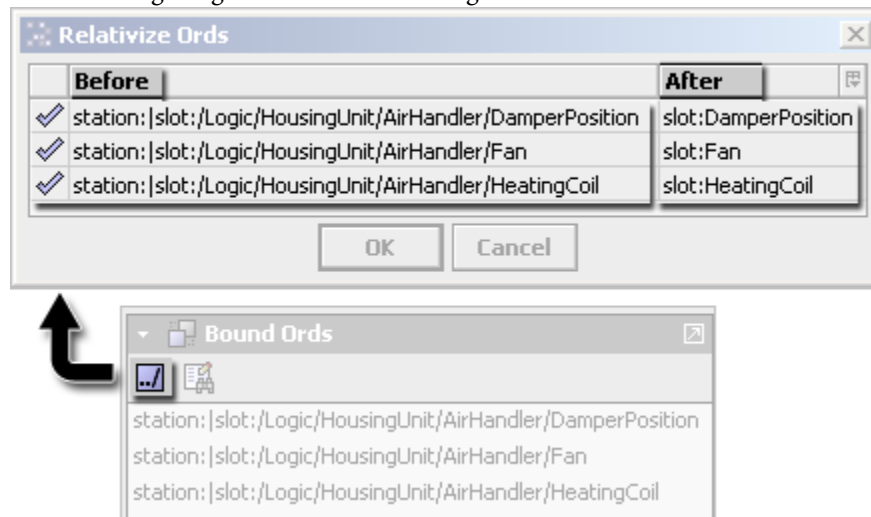
## Relativize absolute ORDs

- Step 1. Open an existing Px view that contains one or more widgets with bindings that have absolute ORDs.
- Step 2. In Edit mode, in the **Bound Ords** area confirm that there are absolute ORDs.

**TIP:** An absolute ORD is relative to the station and shows the entire file path from the station's **Config** node.

- Step 3. In the **Bound Ords** area, click the **Relativize Ords** button.  
The **Relativize Ords** dialog box displays, listing all of the ORDs that can be relativized.
- Step 4. Click **OK**.  
The **Bound Ords** area, you can see the shortened file path of each relativized ORD. The ORDs are relative to the current parent ORD.

The following image illustrates relativizing absolute ORDs:



## Use the zooming options

Starting in AX-3.8, several zooming options are available in Px Editor.

### Zoom-in on a Px Page

#### Prerequisites:

- Target platform running NiagaraAX 3.8

Increase the magnification level of your Px page.

- Step 1. In Px Editor, open an existing Px View containing at least one image.
- Step 2. Click **Px Viewer Toggle View/Edit Mode** to switch to PxEdit mode.
- Step 3. Confirm current zoom level, shown in the status bar in the lower right corner, is x1.0 (100%). If not, click the **Reset Zoom** button.
- Step 4. Click the **Zoom In** button (🔍) as needed, increasing the view x0.2 (20%) with each click.

The Px Editor display zooms-in on the canvas pane. The close-up view allows for greater precision in positioning widgets.

---

**NOTE:** The maximum magnification level is x10.0 (1000%). Also, the zoom level in a Px page is remembered when navigating from page to page.

---

### **Zoom-out on a Px Page**

#### **Prerequisites:**


- Target platform running NiagaraAX 3.8

Decrease the magnification level of your Px page

Step 1. In Px Editor, open an existing Px View containing at least one image.

Step 2. Click **Px Viewer Toggle View/Edit Mode** to switch to PxEdit mode.

Step 3. Confirm current zoom level, shown in the status bar in the lower right corner, is x1.0 (100%). If not, click the **Reset Zoom** button.

Step 4. Click the **Zoom Out** button () as needed, decreasing magnification by x0.2 (20%) with each click.

The Px Editor display zooms-out on the canvas pane displaying more of the page at a reduced size. This is useful when working with large, complex layouts.

---

**NOTE:** The minimum magnification level is x0.2. Also, the zoom level in a Px page is remembered when navigating from page to page.

---

### **Reset zoom on Px Page**

#### **Prerequisites:**

- Target platform running NiagaraAX-3.8
- Existing Px page that has an increased zoom level

Reset the zoom level to x1.0 (100%).

Step 1. Click the **Reset Zoom** button ()

The canvas pane magnification resets to x1.0 (100%), displaying the Px page in actual size.

## **Workflow for using the PxInclude Widget and ORD Variables**

The PxInclude Widget provides a way for you to embed a single Px file in another Px file. The following is a general workflow for using the widget.

Step 1. Create (or open an existing) Px file that you want to use as an included or child Px file. This is the file that may be embedded in parent Px files using the **PxInclude** widget.

Step 2. Design the child Px file, as desired. If you are using any bound variables, consider whether or not to use an ORD variable for the ord property value of any bindings that you may want to assign only at the parent Px file level.

Step 3. Create any ORD variables using the `$(var)` syntax.

Step 4. Save the child Px file in the desired location.

Step 5. Create (or open an existing) parent Px file. This is the file that uses the **PxInclude** widget to embed the child Px file.

Step 6. Add a **PxInclude** widget to the parent Px file using one of the following methods:

- Drag (cut and paste) the widget from a palette and edit the ord property to add the child Px file.
- Drag (cut and paste) the child Px file from the workbench Nav tree to add the widget to the parent Px file with the child Px file already defined in the PxInclude ord property.
- Drag (cut and paste) a component from the Nav tree to initiate the **Make Widget Wizard**. Use the Make Widget Wizard to select the child Px file and choose which variables to bind.

Step 7. Select the **PxInclude** widget and edit the following properties, as required:

- enabled, layout, and visible - leave set to default values.
- ord - click the value field and define or browser to select the child Px file. If you used the **Make Widget wizard**, this field may already be defined.
- variables - click the value field to open the **Variables** dialog box. Use the editor field to define an ORD value for each variable that appears. Notice the After column to verify the correct ORD path for each variable.

Step 8. Click **OK** to close the Variables dialog box.

Step 9. Save the parent Px file and attach it as a view to a component, if desired.

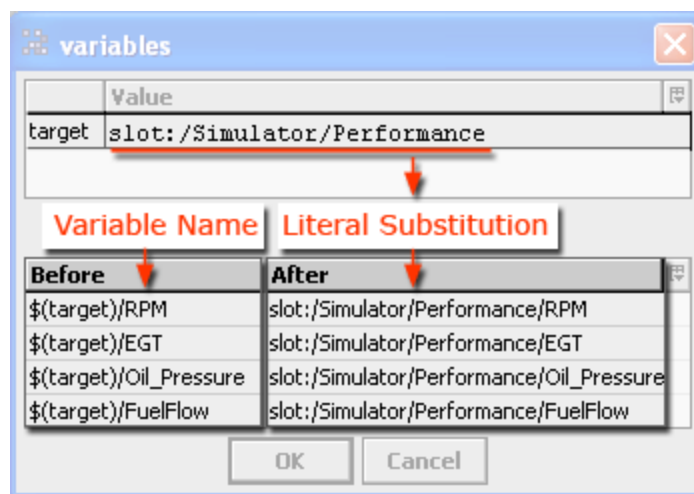
Step 10. Verify that the variable ORD paths are set correctly by viewing the parent Px file from the component view (if used).

### **Using a PxInclude Widget with a single ORD variable (an example)**

The following example illustrates using a PxInclude widget with a single ORD variable.

The figure below shows an example of the **Variables** dialog box with one variable assigned.

**NOTE:** The Variables dialog box displays when you double-click on the “variables” field in a PxInclude widget. This dialog box has an editor field that allows you to assign literal text strings to replace each variable that has been set in the child Px file.



Note the following about this example:

- The top half of the dialog box comprises an "editor" field. This is where you can type a text string in the "Value" column to substitute for the variable.
- This example shows a single variable that is used in four instances. Each instance of the variable ORD has a different point appended to the variable.



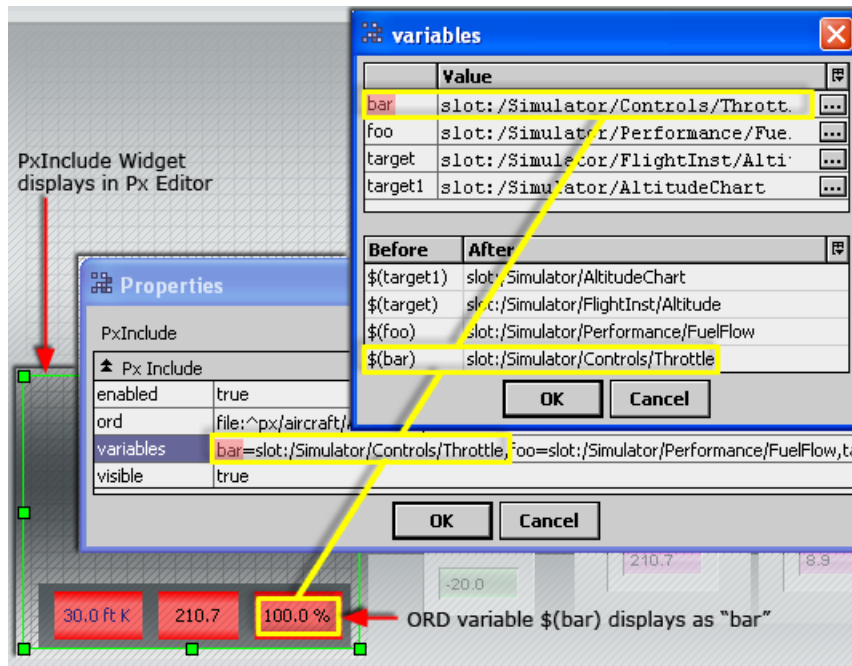
- The bottom-left area of the dialog box displays each instance of the variable, as defined in the child Px file, in a *Before* column.
- The bottom-right area of the dialog box displays the results of any edits entered in the editor field in an *After* column. This is the literal ord value that is used in the Px file.

See also, [Embed Px files with ORD variables using PxInclude, page 13](#)

### Using PxInclude Widgets with multiple ORD variables (an example)

This example illustrates how PxIncludes can use more than one ORD variable.

The figure below shows a Px Editor view of a Px file. This Px file has a **PxInclude Widget** with multiple variables defined, as shown in the widget's **Properties**, and **variables** dialog boxes.



Note the following about this example:

- The **PxInclude widget** is placed in the bottom left corner of the Px Editor view of a parent Px file (displaying 3 of the 4 included ORD variables).
- The ORD variables are defined by typing in the editor field **Value** column of the Variables dialog box. The variable values display in the **After** column in the lower half of the Variables dialog box.
- Defined variables display in the **PxInclude Properties** dialog box (or **Properties** palette) with multiple variables displaying in the **variables** property value field separated by commas.
- Values display in the **PxInclude** widget area only after the variable is defined (using the **Variables** dialog box).

See also, [Embed Px files with ORD variables using PxInclude, page 13](#)

### Use the visible property in Hx profile (an example)

An example of binding and using the **visible** property is shown in the following illustration.

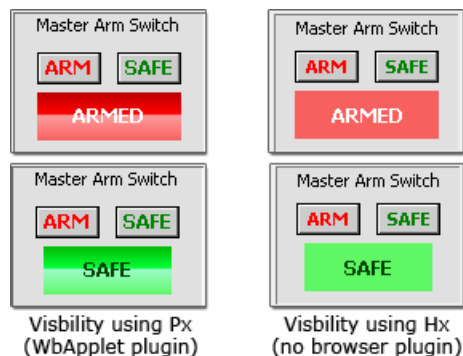
↑ Label	
background	(gradient)
blink	false
enabled	true
font	bold 12pt Tahoma
foreground	#004000
halign	Center
image	null
text	SAFE
textIconGap	4.0
textToIconAlign	Right
valign	Center
visible	station:/slot:/Simulator/MasterArm/out/value
↑ Value Binding	
ord	station:/slot:/Simulator/MasterArm/out/value

The example above shows the **visible** property animated using a value binding. The value binding is linked to a writable point that passes its `true` or `false` value to the visible property so that it is visible (`true`) or hidden (`false`).

Note the following visibility characteristics:

- The **visible** property functions in Hx exactly as it functions in Px.
- Direct children of Tabbed panes cannot have a hidden visibility (`visible = false`).
- Direct children of Scroll panes cannot have a hidden visibility (`visible = false`).
- The **visible** property updates every 5 seconds.

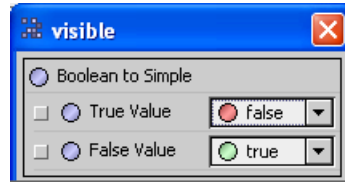
The following illustration gives a comparison of how a simple use of the **visible** property displays in a browser using the WbApplet and using Hx (no applet).



Note the following about this example:

- The **ARM** and **SAFE** buttons write to a boolean point.
- The **ARMED** and **SAFE** labels have a **visible** property that is bound to and animated by the value of the boolean point.
- Clicking the **ARMED** button sets the boolean point value to `"true"`
- Clicking the **SAFE** button sets the boolean point value to `"false"`
- The `"true"` boolean point value sets the **ARMED** label **visible** property to `"true"` and the **SAFE** label **visible** property to `"false"` as defined by their respective **visible** properties using the visible dialog box.
- The `"false"` boolean point value sets the **ARMED** label **visible** property to `"false"` and the **SAFE** label **visible** property to `"true"` as defined by their respective **visible** properties using the visible dialog box.

- The behavior of the **visible** property is the same in both Px (WbApplet) and Hx rendered views in the browser.



## Customizing the login screen

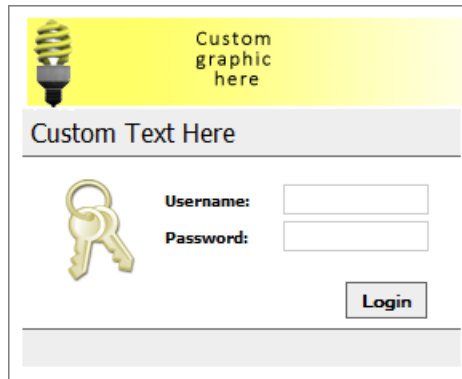
Instructions for customizing the login screen with your own logo and text.

### Prerequisites:

- The logo image that you plan to use must be placed in the file system of the station
- Open connection to your station

### Create a customized login screen

- Step 1. Drag and drop your logo image file from My File System to the Files folder of the Station.
- Step 2. Expand **Config** > **Services**, right-click on the **WebService** node and select **Views** > **Slot Sheet**
- Step 3. Right-click under the current slots and select **Add Slot** from the popup menu.
- Step 4. Enter `logo` as the name and `baja Ord` as the type and click **OK**.
- Step 5. Right-click on the **WebService** node and select **Views** > **Property Sheet**
- Step 6. Scroll down to the **logo** slot and click the down arrow next to the folder icon and select **File Ord Chooser**.
- Step 7. Select the image that you want to use as the logo on the login screen and click **Save**.
- Step 8. Right-click on the **Config** node of the station and select **Views** > **Slot Sheet**
- Step 9. Right-click under the slots and select **Add Slot** from the popup menu.
- Step 10. Enter `displayName` as the name and `baja Format` as the type and click **OK**.
- Step 11. Right-click on the **Config** node and select **Views** > **Property Sheet**
- Step 12. In the **Display Name** field, enter the text that you want to display on the login screen, for example: `Custom Text Here`.  
For example, "Custom Text Here"
- Step 13. Click the **Save** icon in the Workbench toolbar.
- Step 14. Restart the Station and login through a browser window (<http://localhost/login>) to see your customized login screen:




## Apply visual styles to Px views

You can apply visual styles to your Px views in a couple of ways. One method is to configure Px properties. However, these are instance properties, meaning that you must define and configure the properties on every Px page. An alternative is to use Workbench themes to apply styles globally, which means all Px files reference a single properties file, the NSS (Niagara Style Sheet). To learn more, see “About Workbench Themes” in the *Niagara User Guide*.



### Using Px Properties

Typically you work with Px properties in the following ways:


- **Create Px Properties**

Click the **Add New**  button to open the **Add** dialog box. In the **Add** dialog box, you can name the property and choose property types from drop down lists.


- **Edit Px Properties**

In the Properties palette Value column, click on the value **select**  and **edit**  buttons to edit a selected existing property values.

- **Delete Px Properties**

Click the **Delete**  button to delete a selected property.

- **Link Px Properties**


With a Px object selected, right-click on the desired object property and select the **Link**  menu item from the popup menu. Under the "**Link**" submenu, select the property item to apply.

---

**NOTE:** The popup menu is context sensitive. Px Properties that are not appropriate for the selected object property are not available (dimmed).

---

- **Unlink Px Properties**

With a Px object selected, right-click on the linked property and select the **Unlink**  menu item from the popup menu.

## Build navigation files

The tasks in this section are commonly associated with setting up and working with the Nav files in NiagaraAX.

### ***Creating new nav files***

You can have more than one nav file associated with a station. Nav files must reside on the file system, not in the station database.

---

**NOTE:** An efficient method for managing nav files is to create a “nav” folder on the file system to hold all your nav files.

---

For more information, refer to [About the Nav file, page 104](#) and [About the Nav File Editor, page 106](#).

Step 1. In the Nav tree, right-click on the **File system** node.

Step 2. In the **popup** menu, click **New > NavFile.nav**.

The **File Name** dialog box displays with a default file name

Step 3. In the **File Name** dialog box, type a name for the new nav file and click **OK**.

The File Name dialog box disappears and the new nav file appears in the Nav tree.

### ***Deleting nav files***

Step 1. In the Nav tree, right-click on the nav file that you want to delete.

Step 2. In the **popup** menu, click **Delete**.

---

**NOTE:** An alternative method is to select the nav file in the Nav tree and press the **Delete** key on the keyboard.

---

The nav file is deleted.

### ***Renaming nav files***

Step 1. In the Nav tree, right-click on the file system node.

Step 2. In the **popup** menu, click **Rename**.

Step 3. In the **Rename** dialog box, type a new name for the nav file in the text field and click **OK**.

The nav file is renamed.

### ***Editing nav files***

You can edit nav files in the Text Editor or in the Nav File Editor. The procedures in this section cover opening nav files, adding/deleting nodes, editing node hierarchy and editing nodes in a nav file. For more information, see [About the Nav File Editor, page 106](#).

### **Open nav files**

The nav file resides on the file system, not in the station database.

---

**TIP:** Keeping all of your station nav files in a “nav” folder simplifies file management.

---

Step 1. In the Nav tree, find the nav file that you want to open and do one of the following:

- Double-click on the nav file, or
- Right-click on the nav file and select **View > NavFileEditor** from the popup menu.

The **Nav File Editor** opens displaying the file in the **Result Tree** pane.

Step 2. Edit the nav file, as desired.

Step 3. Click the **Save** icon on the toolbar or select **File > Save** to save your changes.

### Add nodes in nav files

Step 1. Open the desired nav file in the Nav File Editor (refer to [Open nav files, page 43](#), if necessary).

Step 2. Add a node to the **Result Pane** using any of the following methods:

- **Copy and paste**

Copy a component or Px file from the Nav side bar and paste it into the Result Tree pane. The new node appears in the Result Tree as a child node of the node that you target when pasting.

- **Drag and drop (from the Nav side bar)**

Drag a component or Px file from the Nav side bar and drop it onto the Result Tree pane. A new node appears in the Result Tree as a child node of the node that you target when dropping it.

- **Popup menu**

Right-click on the component that you want to add and select Copy from the popup menu. Right-click the desired target area in the Result Tree, and select Paste from the popup menu. A new node appears in the Result Tree as a child node of the node that you target when pasting.

- **New button**

Click the **New** button on the bottom of the Nav File Editor pane. The **New Node** dialog box appears. Edit the Display Name, Target Ord, and Icon fields, as desired and click **OK**. A new node appears in the Result Tree as a child node of the node that is selected.

- **Drag and drop (from the Source Tree)**

Toggle the **Show Components** button and the **Show Files** button to display the desired objects in the Source Tree pane. Drag the desired component or Px file from the Source Tree pane and drop it onto the Result Tree pane at the desired location. A new node appears in the Result Tree as a child node of the node that you target.

Step 3. Click the **Save** icon on the toolbar or select **File > Save** to save your changes.

### Delete nodes in nav files

Step 1. Open the desired nav file in the **Nav File Editor** (refer to [Open nav files, page 43](#), if necessary).

Step 2. In the Result Tree pane, locate the node that you want to delete and do one of the following:

- Select the node that you want to delete and click **Delete**, or
- Select the node that you want to delete and press Delete on the keyboard, or
- Right-click on the node that you want to delete and select **Delete** from the **popup** menu.

### Edit node hierarchy

Node hierarchy in a nav file is the position of parent and child nodes in a tree structure.

Step 1. Open the desired nav file in the **Nav File Editor** (refer to [Open nav files, page 43](#), if necessary).

Step 2. In the Nav tree, locate the nav file that you want to edit and do one of the following:

- Drag and drop the desired tree node from one location to another in the tree.
- Select one or more child nodes in the tree and use the **Move Up** or **Move Down** buttons to raise or lower the node in its parent node.

---

**NOTE:** The **Move Up** or **Move Down** buttons move nodes only within their parent node. To move a node to a different parent, you must cut and paste the node.

---

The **Nav File Editor** displays the new node location in the Result Tree pane.

Step 3. Click the **Save** icon on the toolbar or select **File > Save** to save your changes.

### **Edit node properties**

Step 1. Open the desired nav file in the **Nav File Editor** (refer to [Open nav files, page 43](#), if necessary).

Step 2. In the Nav tree, locate the nav file that you want to edit and do one of the following:

- Double-click on the desired node, or
- Select the desired node and click the **Edit** button.
- Right-click on the node file and select **Edit** from the **popup** menu.

Step 3. In the **Edit** dialog box, edit the node display **Name**, **Target Ord**, and **Icon** properties, as desired and click **OK**.

Step 4. Click the **Save** icon on the toolbar or select **File > Save** to save your changes.

## **Generate reports**

The Reporting function in NiagaraAX helps you design, display, and deliver data to online views and to printed pages.

The process of creating and sending a report can be performed in a variety ways. The following example illustrates the basic workflow for the reporting process.

### ***Reporting process workflow***

The following list is a basic workflow for the reporting process.

- **Setup the reporting service**

Add the ReportService and child components (ExportSource and EmailRecipient) from the Reporting palette Services folder.

- **Layout report data in a ComponentGrid.**

Add a ComponentGrid component (from the Reporting palette) onto the desired location in your Nav tree. Using the Grid Editor view, drag and drop points from the Nav tree or use the popup menu or main menu icons to add rows and columns to the ComponentGrid.

- **Create a Px page display of the report**

Using the Px editor, add the ComponentGrid to a ReportPxFile page and design the report layout using ReportWidgets available from the Report palette.

- **Configure ExportSource component**

In the ExportSource property sheet, identify the report Px file and schedule the report delivery.

- **Configure EmailRecipient component**

In the EmailRecipient property sheet, identify the desired recipients and email account to use for delivery of the report.

### **Set up the reporting service**

Add the reporting service and components to the station.

Step 1. In the Palette side bar, open the **report** palette.

Step 2. Expand the **Reporting** folder.

Step 3. Drag and drop the following components **ReportService** and child components (**ExportSource** and **EmailRecipient**) into your station Services node.

**ReportingService** is setup and ready to use.

### **Set up report data in a ComponentGrid**

Add a ComponentGrid component (from the Reporting palette) onto the desired location in your Nav tree. Using the Grid Editor view, drag and drop points from the Nav tree or use the popup menu or main menu icons to add rows and columns to the ComponentGrid.

Step 1. In the Palette side bar, open the **report** palette.

Step 2. Open the Reporting folder in the **report** palette and drag and drop the **Component-Grid** component into the Config node in your station.

Step 3. Right-click on the new **ComponentGrid** component and select **Views > Grid Editor**. The **Grid Editor** view displays.

Step 4. Expand the Nav tree to locate the desired network and expand the network so that you can see the devices in the tree.

Step 5. Click and drag a single device from the Nav tree onto the middle (yellow) pane in the **Grid Editor**.

---

**NOTE:** You must define this template row before adding columns

---

This also adds the same device to the lower pane as one of the defined rows.

Step 6. In the Nav tree, click and drag the same device to the top pane of the **Grid Editor** (no columns defined).

This adds a column definition which defaults to the `displayName` of the device.

Step 7. Again, click and drag the same device in the Nav tree to the top pane of the **Grid Editor**.

This adds a second column definition.

Step 8. Double-click the second column definition in the top pane of the **Grid Editor** to open the **Edit** dialog box. Change the **name** property to "Status" and change the **format** property to `%status%`, and click the **OK** button.

Step 9. In the Nav tree, select all of the other devices that you wish to include in the table, then click and drag them onto the lower pane of the **Grid Editor**.

This adds all of those devices as separate rows of the grid table.

Step 10. Save the **Grid Editor** and change to the **Grid Table** view to see the completed table.

### **Creating A ReportPxFile**

Step 1. Right-click on a container for a Px file and select **New > ReportPxFile.px**



- Step 2. In the Name for New File dialog box, edit the default file name, if desired, and click **OK**.  
The new Px file, pre-loaded with a **ReportPane** widget, appears in the Nav tree.
- Step 3. Double-click on the new **ReportPxFile** to open it in PxViewer.
- Step 4. Switch to Edit mode.
- Step 5. Open the **report** palette, expand the **Reporting** folder and drag a **ComponentGrid** widget to the canvas pane.
- Step 6. In the Make Widget wizard, click **Workbench View** and in the secondary view area select **Grid Editor** and click OK.
- Step 7. Design the report layout using Report Widgets.

### **Configuring ExportSource component**

#### **Prerequisites:**

- An existing ReportPxFile
- Step 1. In the station, expand **Config > Services > ReportServices** and double-click on the **ExportSource** component to open the Property Sheet view.
  - Step 2. In the **Source** slot, click on the icon to launch the ReportPxFile using the **Export Source wizard**.
  - Step 3. In the **Export Source** wizard, click the folder icon and browse to your **ReportPxFile** and select it click **Open** to close the dialog box.
  - Step 4. In the **Export Source** wizard, click **Next** to continue and **Finish**.
  - Step 5. In the Property Sheet view, expand the **Schedule** slot to schedule the report delivery. Select from the following options:
    - Trigger Mode (Manual, Daily, or Interval)
    - Time Of Day
    - Randomization
    - Days of Week
    - Last Trigger
  - Step 6. Click **Save**.

You have configured the report source and delivery schedule.

### **Configuring EmailRecipient component**

Specify where a report is to be emailed.

#### **Prerequisites:**

- An existing ReportPxFile
- Step 1. In the station, expand **Config > Services > ReportServices** and double-click on the **EmailRecipient** component to open the Property Sheet view.
  - Step 2. Enter the name and email address of recipient(s).
  - Step 3. Click **Save**.



# CHAPTER 3 Px GRAPHICS REFERENCE

## TOPICS COVERED IN THIS CHAPTER

- [About Px views](#)
- [About Px files](#)
- [About Px viewer](#)
- [About Px Editor](#)
- [About the View Source XML dialog box](#)
- [About Widgets](#)
- [About developing for portability](#)
- [About the PxInclude Widget](#)
- [About Animating Graphics](#)
- [About types of Px target media](#)
- [About profiles](#)
- [About the Px Editor workflow](#)
- [About the Make Widget wizard](#)
- [About the Nav file](#)
- [About the Nav File Editor](#)
- [About Reporting](#)
- [About Mobile](#)

This section describes the basic presentation concepts and the graphics presentation tools that are available in NiagaraAX Workbench for AX-3.7 and AX-3.8.

## About Px views

A Px view is a custom graphical view of control logic that you define in a Px file. The view provides a visualization of information in a dynamic graphical format.

A Px view is a custom graphical view that you define in a Px file. The purpose of a Px view is to provide a visualization of information in a rich, dynamic format that is *easy to create and to edit*. By using the Px Editor to build Px files, you can create the desired visualization of your control logic without having software programming skills. When attached to a component, the Px view becomes the default view for the component.

Looking at an object's Px view in the property sheet (where it may be edited) helps illustrate what a Px view is. A Px view is comprised of the following parts.

- **Icon**

Assign or change the icon file that appears to the left of the Px view display name. After assigning this icon to the view, it appears in the view selector menu, the Nav tree side bar, and in the property sheet view.

Icon graphics must be linked into your view from a module (for example, use the following path to find a "folder" icon: "module://icons/x16/folder.png"). Icon graphics are not supported outside of modules.

- **Px File**

Assign the Px file that is associated with active Px view. See [About Px files, page 53](#) for more about Px file.

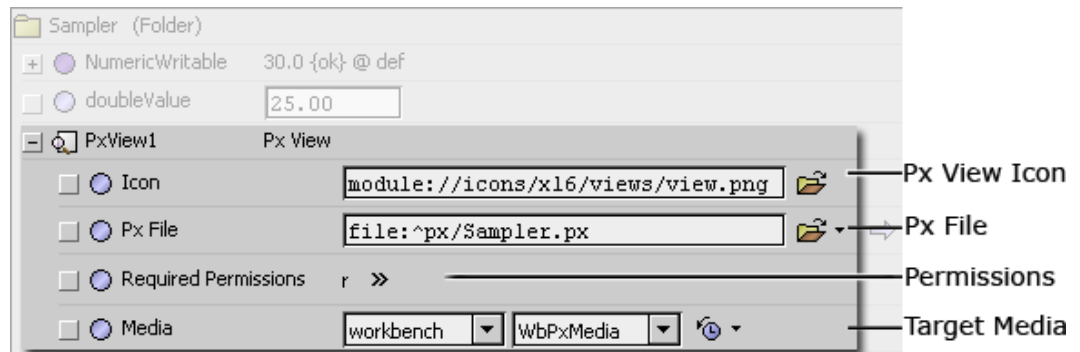
- **Required Permissions**

Assign or edit the Operator and Admin permissions that are applicable when the view is active: Read (r), Write (w), Invoke (I).

- **Media**

Edit the target media of the Px view: `workbench` or `html`. See [About types of Px target media, page 92](#) for details about target media types.

**Figure 1. Px view in a component property sheet**



**Px views as slot properties**

The Px view is attached to a component as a dynamic slot property of the type “baja : PxView.” You can add new Px views directly in the slot sheet view.

For more details, see the section “To add a new slot” in the .

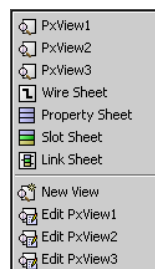
The following graphic shows three Px views in a single component slot sheet.

**Figure 2. Three Px views as slot properties**

Slot	#	Name	Display Name	Definition	Flags	Type	Facets
Property 0	0	NumericWritable	NumericWritable	Dynamic		control:NumericWritable	
Property 1	1	doubleValue	doubleValue	Dynamic		baja:Double	
Property 2	2	PxView1	PxView1	Dynamic	o	baja:PxView	
Property 3	3	PxView2	PxView2	Dynamic	o	baja:PxView	
Property 4	4	PxView3	PxView3	Dynamic	o	baja:PxView	

The Px view highest on the slot sheet is the default view. You can reorder Px views to change the default Px view of a component. For an active component, all Px views (if any) appear in the view selector.

**Figure 3. Three Px views in the view selector**



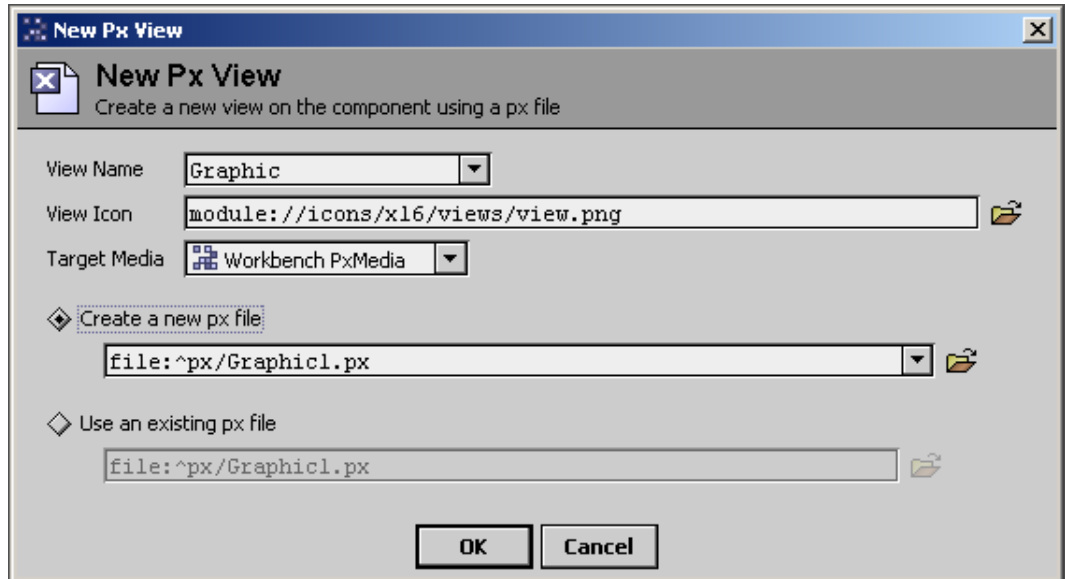
**About the Px View wizard**

You can create and delete Px views using the Slot Sheet view, as described in “Px views as slot properties”. However, the easiest way to create a new Px view on a component is to use the **New Px View** wizard. Start the **New Px View** wizard by selecting **New View** from the **popup** menu or from the **View Selector** menu.

You can create and delete Px views using the Slot Sheet view, as described in “Px views as slot properties”. However, the easiest way to create a new Px view on a component is to use the **New Px View** wizard. Start the **New Px View** wizard by selecting **New View** from the **popup** menu or from the **View Selector** menu.

This wizard, shown below, is a single dialog box that allows you to assign the view name, view icon, and target media properties, as described in [“About Px views”, page 49](#). The wizard creates a new Px file and assigns it a default name unless you specify a name. You may choose to assign an existing Px file to the view instead of creating a new one.

**Figure 4. New Px View wizard**



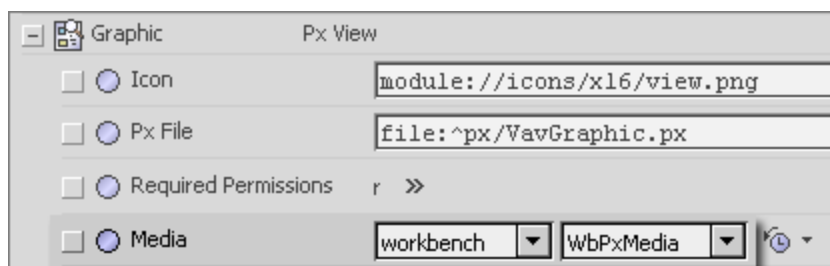
After completing the fields in the **New Px View** wizard, click the **OK** button to finish. The new Px view is displayed in the Px Editor. If you choose to have the wizard create a new Px file, it will be a simple default file, as described in [“About Px files”, page 53](#).

### Choosing a Px Media type

You can design Px files for a specific primary media type and indicate the media type on the Px view property sheet, as shown below.

Choosing a media type has specific benefits that help you get started when you first create a new view on a component and while you are working in the Px Editor.

**Figure 5. Specifying the target media**



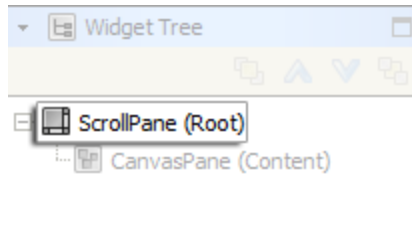
**NOTE:** Media values affect the initial contents of a Px file when you create it. Changing a view's **Media** type (on a component property sheet) after the Px file is created does not actually change the Px file.

In addition to the default, `WbPxMedia` type, you can select one of the following when creating a new view on a component:

- **HxPxMedia**

Choosing this media type provides you with a Px file that has a ScrollPane with a CanvasPane. With this property value set, the Px Editor can warn you, as you design your Px file, if you choose a widget that is not supported by Hx media.

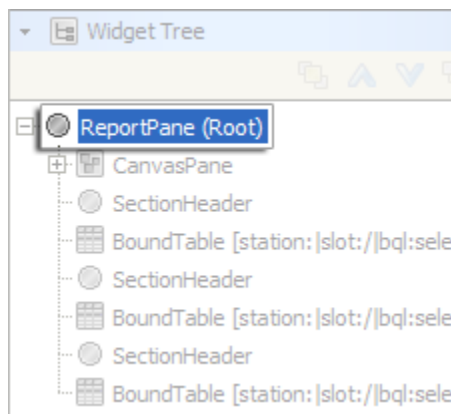
**Figure 6. HxPxMedia creates Px page with Scroll and Canvas Panes**



- **ReportPxMedia**

Assigning this media type when creating a new Px page, creates a new Px file with a Report Pane containing a timestamp and page numbering at the root of the Px file.

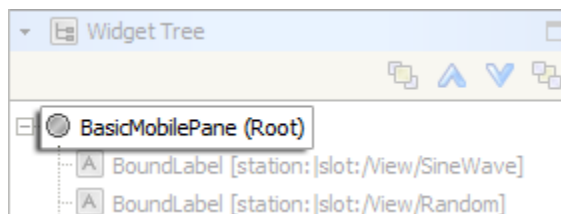
**Figure 7. ReportPxMedia creates Px page with Report Pane**



- **MobilePxMedia**

You must use a BasicMobilePane at the root of your Px page so it will display correctly on a mobile device. Assigning a MobilePxMedia type when creating a new Px page, causes the new page to come with a BasicMobilePane at the root of the Px file, as shown in the following figure.

**Figure 8. MobilePxMedia creates Px page with BasicMobile Pane**



---

**NOTE:** Starting in AX-3.8, setting the CanvasPane `scaleMode` property to `FitRatio` will cause your Px page to automatically to scale to fit various device display sizes.

---

## About Px files

Px file defines the content and presentation of a Px view.

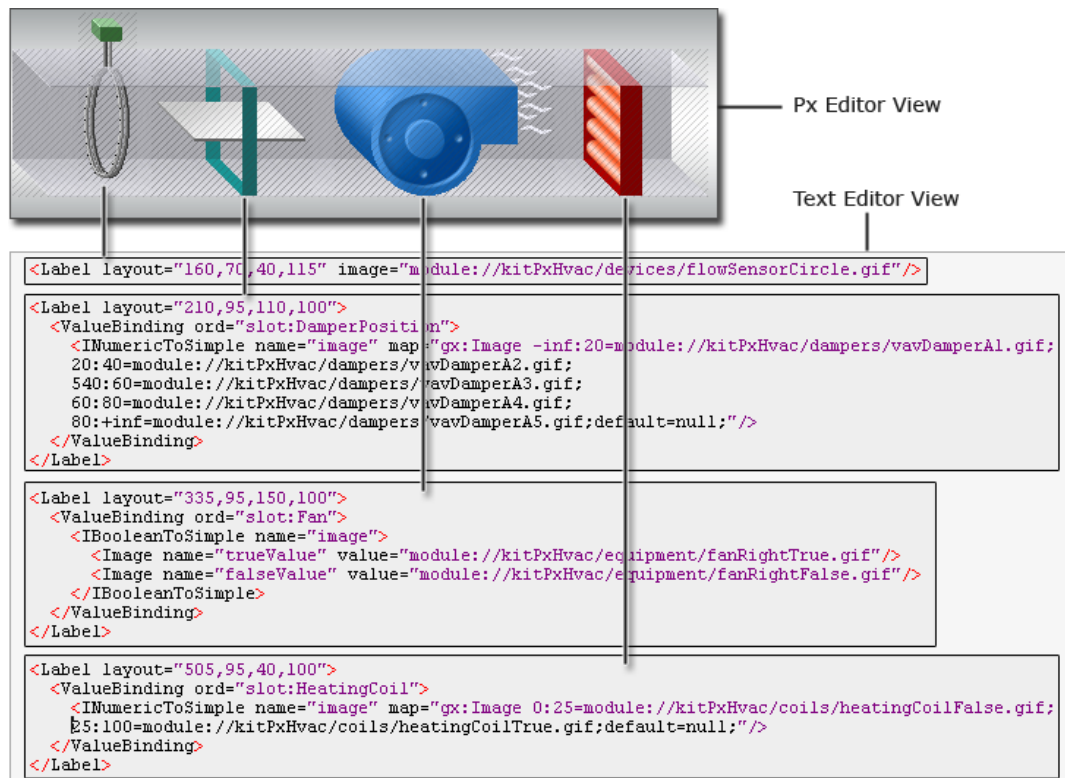
The Px file is a special XML file that describes the components in a database and can be any collection of components, up to a complete database. All views (wire sheet view, property sheet view, category sheet view, and so on) can be used on components in a Px file. This means that a Px view can be used to provide a complete variety of options in the development of dynamic user interfaces.

The contents of a very basic Px file is shown in the figure below as it appears in the Text Editor. This file simply contains an empty canvas pane nested in a scroll pane.

**Figure 9. Basic default Px file in Text Editor View**

```
<?xml version="1.0" encoding="UTF-8"?>
<px version='1.0' media='html:HtmlPxMedia'>
  <import>
    <module name='bajau' />
  </import>
  <content>
    <ScrollPane>
      <CanvasPane name="content" viewSize="500,400"/>
    </ScrollPane>
  </content>
</px>
```

As you add more objects to the file, (typically using the Px Editor) the file looks more like the snippet shown here. The graphic components, value bindings, and their container elements and attributes are all referenced in the Px file.

**Figure 10. Px file in Text Editor View and in Px Editor View**

The elements in the Px file are also graphically represented in the Widget Tree side bar pane.

**NOTE:** Two types of Px files are available. The standard Px file (includes a scroll pane at the root with a canvas pane) or the ReportPxFile (contains a report pane at the root).

- For information refer to the Px Editor.
- For information refer to the Widget Tree side bar.
- For more information refer to the Workbench Text Editor

### Shared Px files

A Px file may be used as part of one or more Px views. Editing a shared Px file affects all views that use it.

Since the bindings within a Px file are always resolved relative to the current ORD, you can reuse the same Px file across multiple components by specifying bindings with relative ORDs. This allows you to create a single presentation and use it in views that are assigned to components that can use identical graphic layout. The obvious advantage of this file sharing is that you only need to create and edit one Px file for many views, thus saving time and ensuring consistency among similar applications. However, it requires that you understand the following caution:

**CAUTION:** Editing a Px file affects all views that use that particular Px file. Any time that you view a component in the Px Editor you have its Px file open for editing. If the Px file is shared with other Px views, then all of those Px Views are changed.

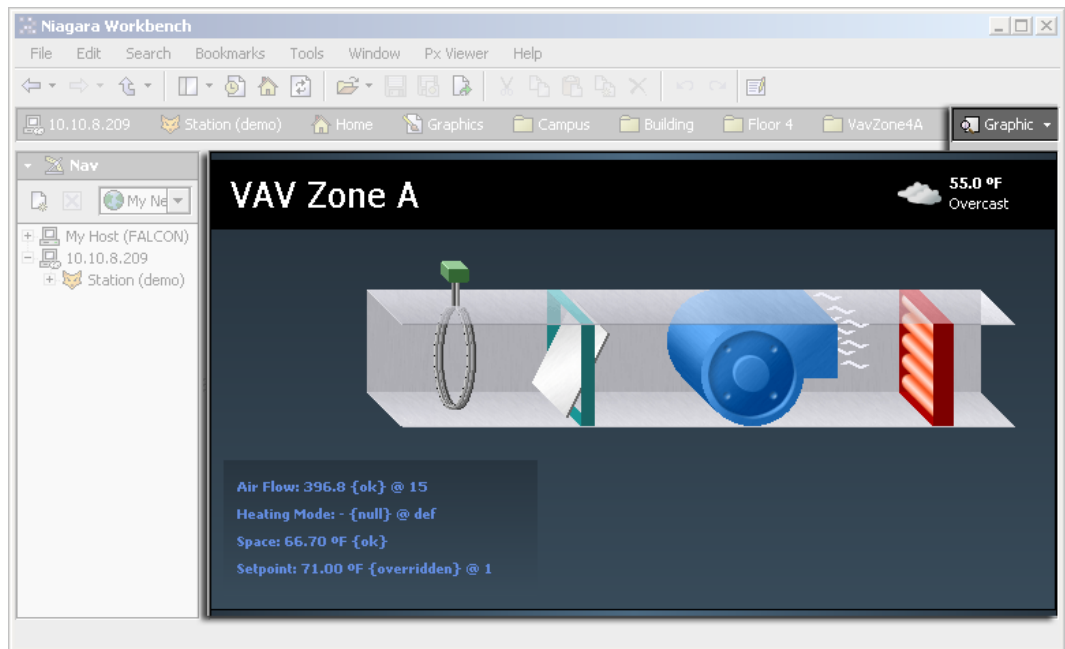


## About Px viewer

The Px viewer presents Px in a non-editable display. The Px viewer allows the user (with required permissions) to view information and invoke actions on controls in the display.

Any time you select a Px view from the right-click menu or from the View Selector menu, the Px viewer displays the active component's view, as shown below. Use the Px editor to edit the Px file (as described in [About Px Editor, page 55](#)).

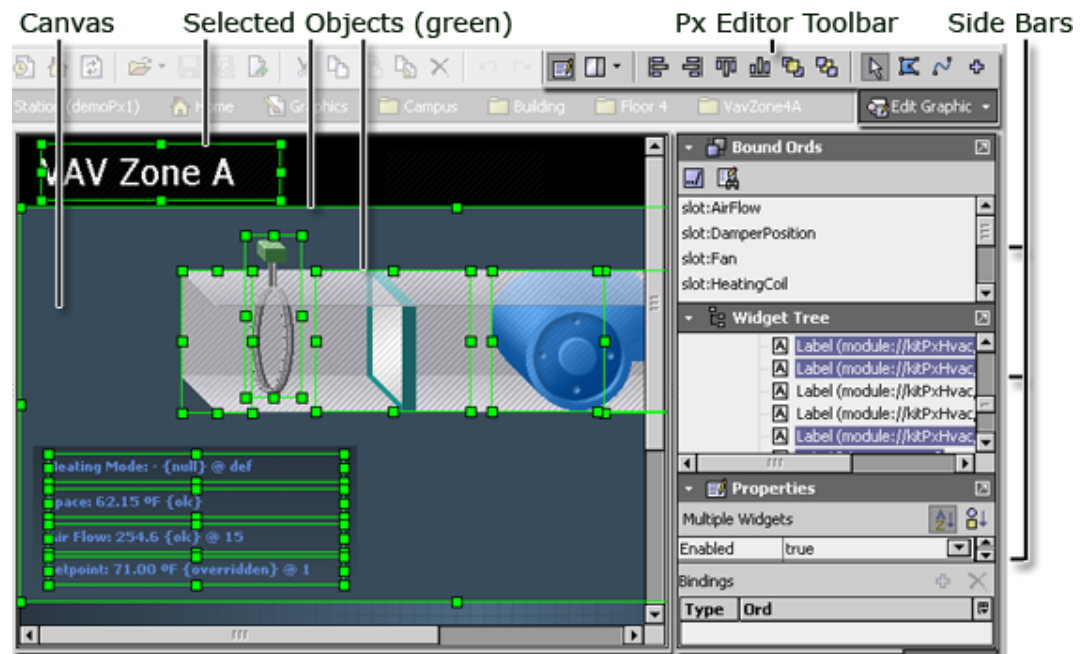
**Figure 11. Px Viewer**



## About Px Editor

When you select the Px Editor view of a component, or of a Px file, the Px Editor displays the Px file in the Workbench view pane.

The figure below shows a Px View of a component with several items selected.

**Figure 12. Px Editor**

The default view of the Px Editor comprises three main areas:

- **Canvas**

The largest area of the Px Editor (by default) is the canvas pane. Typically, you place widgets on the canvas and edit them using one or more of the three side bars and additional dialog boxes. If you want to change the default size of the canvas pane that appears in new Px files, then edit the canvas pane that is in the "/workbench/newfiles/PxFile.px" file (under your NiagaraAX installation ("My Host > My File System > Sys Home") directory. For more information, see [About Widgets, page 65](#).

- **Side bar pane**

The Px Editor side bar pane appears on the far right side of the view pane only when the Px Editor is active. Use the Px Editor toolbar menu to hide or display individual side bars or to show or hide the Px Editor side bar pane. Refer to the *NiagaraAX User Guide* for more information about individual side bars.

- **Px Editor toolbar**

The Px Editor toolbar displays above the view pane when the Px Editor is active. Refer to *NiagaraAX User Guide* for more information about individual Px Editor toolbar icons.

### **About Px Editor Canvas**

The Canvas defines the visual boundaries of the graphic page that you produce in the Px Editor. The Canvas is also your work area for previewing the Px file as you develop it using the tools in the Px Editor.

The Canvas, shown below, provides a view of the widgets as you add them and bind data to them. Most of the time, the Canvas provides a live view of any widgets that are added—without having to return to the Px Viewer. However, some graphic features may only appear in the Px Viewer. For more information, see [About Px viewer, page 55](#).

The Canvas has the following optional work aids:

You can change the default settings of many of the Px Editor display features in the Px Editor view of the Options dialog box (select **Tools > Options** from the Workbench menu bar).

- **Grid**

This is a visual aid for graphical alignment. The grid lines display vertical and horizontal lines as well as define the visible area of the page. Toggle the grid on or off using the Px Editor menu.

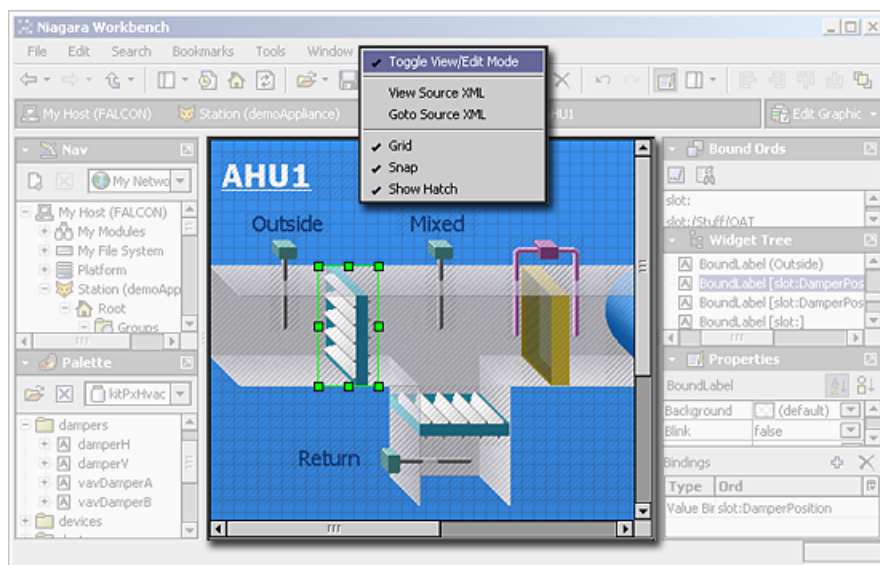
- **Show Hatch**

Hatching is an area of light-gray diagonal lines that define the boundaries of items that are placed on the Px Editor Canvas. Toggle the hatching on or off using the Px Editor menu.

- **Snap**

This feature is an aid to precise placement when you are using the mouse to drag an object to a particular location. Snap pulls the item to the nearest grid line or nearest object for a matching vertical or horizontal alignment. Toggle Snap on or off using the Px Editor menu.

**Figure 13. Px Editor Canvas**

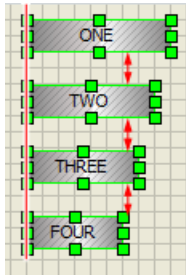


### About alignment and distribution tools

You can use Px Editor tools to help you align and distribute objects in the Px Editor. The align actions are available from the Px Editor toolbar and both align and distributed features are available directly from the right-click popup menu.

- **Distribute**

If you select three or more widgets in the Px editor, you can use the **Distribute** feature to move the widgets so that the empty space between the objects is distributed evenly. You can choose to distribute the widgets either horizontally or vertically. To use this feature, select the desired widgets and right click on one of them. Select **Distribute > Vertical** or **Distribute > Horizontal** to invoke the distribution action.

**Figure 14. Four labels left-aligned, vertically-distributed**

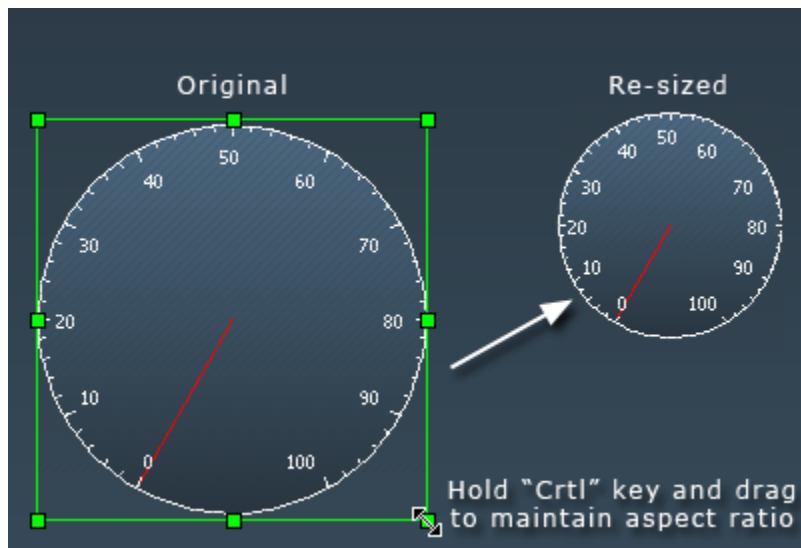
- **Align**

If you select two or more widgets in the Px editor, you can use the **Align** feature to move the widgets so that the left edge, right edge, or center line of the objects align. To use this feature, select the desired objects and right-click over one of them. Select **Align** from the popup menu and choose the desired align option.

### **About Widget sizing in Px Editor**

In order to resize a widget on the canvas pane you must be in Edit mode. Click and drag on one of the widget bounding box handles, as shown below, to resize the widget as desired.

You can easily maintain the aspect ratio of the widget (maintain the widget's height-versus-width proportion) during resizing by pressing the Ctrl key before selecting and dragging the widget bounding box.

**Figure 15. Resizing a widget while maintaining aspect ratio**

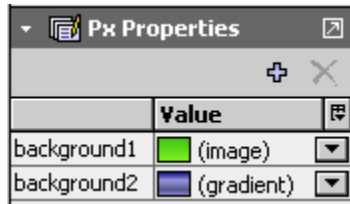
**NOTE:** In AX-3.8 Workbench supports the use of SVG images which retain image quality when scaled up or down in size. For more details, refer to [About SVG support, page 62](#).

### **About Px Properties**

You can use Px Properties to facilitate graphic view design in the Px Editor view. Px Properties are entities based on a Px file markup convention that provides stylesheet-like control for properties in a Px file.

You specify Px properties using a Px Properties palette and a widget's context-sensitive popup menu to "link" properties to Px objects in the appropriate object properties field. The following figure shows an example of two Px properties that are defined in the Px Properties palette.

**Figure 16. Px Properties palette with two defined properties**

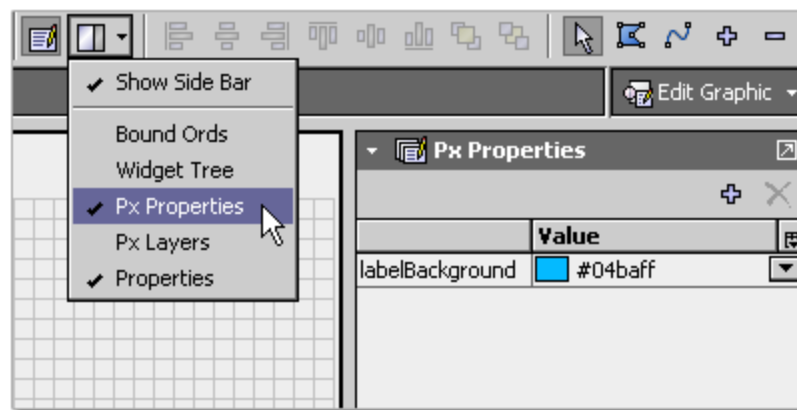


Px Properties can set values for widget properties such as font family, font size, color, background image, and others. After you create a property, you link the property to one or more specific widget properties. This gives you a single source for setting all linked property values on a page. You can use these to easily theme and update your graphics (colors, font types, font sizes, etc.) on a single Px view, as needed.

Use the PxEditor Sidebar icon (located on the main menu bar) to display or hide the Px Properties palette in the Px Editor side bar pane.

**NOTE:** Px properties are not "globally" applicable; they work only on the local Px page where they are defined. However, you can copy and paste the content of the <properties> xml elements from one Px file to another using the Text File Editor view.

**Figure 17. Select the Px Properties palette from the Px Editor Side Bars option menu**



## Px Property Concepts

Use Px Properties to easily apply property values to Px file properties. Once defined, property parameters are assigned to or removed from individual Px object properties by Linking or Unlinking.

When you create a Px property, a <properties> tag pair is entered into the Px file code to hold all property definitions. Each property that is created is defined by a set of parameters (such as "name", "type", and "value") and enclosed in a pair of <property> tags.

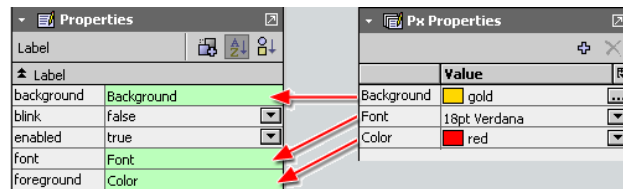
**Figure 18. Px Properties assigned in the Px file code**

```

<properties>
  <property name="Background" type="gx:Brush" value="gold">
  </property>
  <property name="Font" type="gx:Font" value="bold 18pt Verdana">
  </property>
  <property name="Color" type="gx:Brush" value="red">
  </property>
</properties>
<content>
<S: Px Properties
  Applied to Label Object :ent" viewSize="500.0,400.0">
  <BorderPane layout="340,340,150,50" padding="0" border="1.0 inset black">
  <Label name="content" text="Floor 1 VAV 1" font="bold 18pt Verdana"
  foreground="red" background="gold"/>
  </BorderPane>
</CanvasPane>
</ScrollPane>
</content>
  
```

These property parameters are assigned to or removed from individual Px object properties through a process of Linking or Unlinking. When an object property is assigned a Px Property (using the **Link** popup menu item), the property sheet displays the Px Property name in the property value field with a light green background shading.

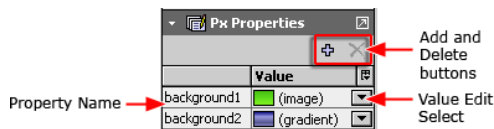
**Figure 19. Linked Px Property Names Display in Property Value Field with Shading**



**About the Px Property palette**


The Px Property palette works just like other Px palettes and has similar controls and displays. The following illustration shows an example Px Properties palette.

**Figure 20. Px Properties Palette**



Note the following about the Px Properties palette:

- Palette controls are located across the title bar and work like other Px palette controls.
- The **Add** button opens the **Add** dialog box for adding a new Px property.
- The **Delete** button deletes any currently selected property.
- Property names are displayed in the left column of the palette table.
- Property values are displayed in the right *Value* column of the table.

- The value select  and edit buttons are located to the right side of each property value, as required.

For more details, see [Using Px Properties, page 42](#).

### About Px Layers

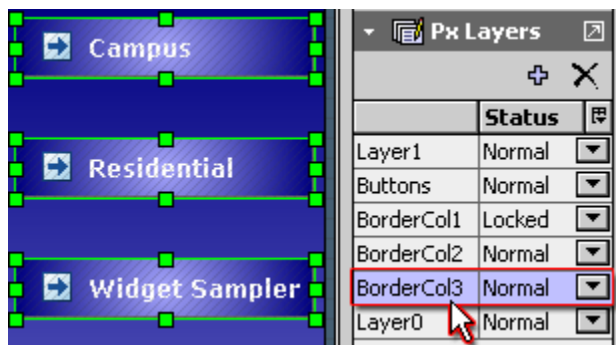
You can use Px Layers to help with graphic design work in the Px Editor view. Px Layers allow you to group objects in the Px Editor view by assigning them to a common “layer”.

The following list describes some of the things you can do with layers.

- **Select all objects assigned to a layer**

You can select all objects in a layer by selecting the desired layer in the Px Layers palette. This saves you from having to individually select each item in the layer. When objects are selected using the Px Layers palette, most Px Editor “edit” actions work the same way as if you had selected the objects individually. For example, when you right-click on a layer in the Px Layers palette, choosing the **Rename** or **Remove** popup menu items renames or deletes the selected layer (not the objects). However, the **Cut**, **Copy**, **Paste**, **Duplicate**, and **Delete** (and other) menu items perform actions on the selected objects, not the layer.

**Figure 21. Selecting a layer**



- **Lock objects assigned to a layer**

You can lock all layer objects by selecting the `Locked` option in the Px Layers palette Status column. Locked objects display normally but you cannot select them on the Px Editor canvas (you can still select them in the Widget Tree). This is convenient for working with stacked or overlapping objects.

---

**NOTE:** Locked object property values cannot be edited and display with a gray background in the Properties palette.

---

- **Hide objects assigned to a layer**

You can hide all layer objects by selecting the `Invisible` value in the Px Layers palette Status column. When objects are invisible you cannot select them on the Px Editor canvas but you can still see and select them in the Widget Tree.

---

**NOTE:** Hidden object property values cannot be edited and display with a gray background in the Properties palette.

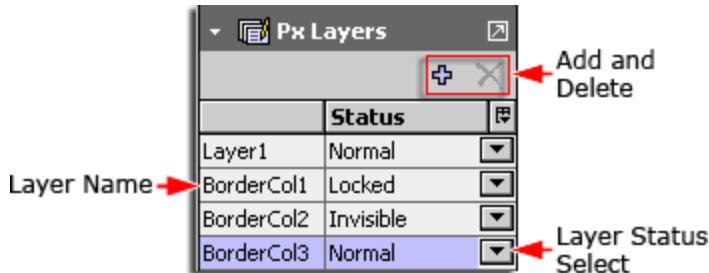
---

Px Layers work using a Px file markup convention that adds a `<layer>` element to the Px file to define each new layer and a `<LayerTag>` element to each object that is assigned to the layer. Normally, you work with Px layers directly in the Px Editor canvas as described in [Group objects using Px Layers, page 14](#).

## About the Px Layers palette

The Px Layers palette works just like other Px palettes and has similar controls and displays. The following illustration shows an example Px Layers palette.

**Figure 22. Px Layers palette**



Note the following about the Px Layers palette:

- Palette controls are located across the title bar and work like other Px palette controls.
- The **Add**



button opens the **Add** dialog box for adding a new Px layer.

- The **Delete** button



deletes any currently selected layer.

- Layer names are displayed in the left column of the palette table.
- Layer status values (Normal, Locked, Invisible) are displayed in the right Status column of the table.
- The value select



button is located to the right side of each layer value.

For more details, see [. , page 14](#)

## About SVG support

Starting in AX 3.8, Workbench supports SVG (Scalable Vector Graphics) rendering. With the `svgBatik` module installed, you can use SVG images in your Px pages just as you would use a GIF, PNG, or JPG image.

SVG images are supported as:

- the image and background properties of a Label or Button
- the image property of a Picture
- the background property of a CanvasPane

The benefit of using an SVG image is that you can scale it up to a larger size without the image quality degrading, becoming pixelated, as happens with other image formats. Also, large SVG images scale down in size, to fit mobile device displays, retaining image quality and legibility. In order to provide this scaling functionality, a new widget called a Picture has been added to the `bajui` palette. You can create a Picture by a number of methods:

- by dragging it from the `bajui` palette



- by dragging an image file from the Nav tree
- or through the right-click menu

A Picture has a `scaleMode` property that you can use to stretch and skew an image to fit within the Picture widget's borders. A Picture can be backed with any image, including JPGs and PNGs, but to get the most benefit out of Picture scaling, SVGs are recommended. For more details, see [Create a scalable image using the Picture widget, page 12](#).

---

**NOTE:** Labels, Buttons, and CanvasPanels support SVG display, but not scaling. These widgets will only display the SVG at its original size.

---

### About SVG rendering in Workbench

In Workbench, SVG images are rendered using the Apache Batik library, packaged in the `svgBatik` module. To keep module size small, only the display features of SVG images are supported, not the interactive features.

How an SVG image is rendered is determined by the program rendering them. In Workbench, SVG images are rendered using the Apache Batik library, packaged in the `svgBatik` module. The capabilities, however, have been reduced in order to keep the module to a reasonable size for JACE installation. For this reason, certain features such as embedded Javascript and mouse events are not supported. In other words, the display features of SVG images are supported, but not the interactive features. CSS- and XML-based animations are supported, but Javascript-based animations are not.

---

**NOTE:** Interactive features of SVG images, such as embedded Javascript-based animations, are not supported.

---

### AX-compatible SVG images

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics that has support for interactivity and animation. All major HTML-5 web browsers support SVG images.

SVG images and behaviors are defined in XML text files. As XML files, SVG images can be created and edited with any text editor, but it is more convenient to create and edit them using a drawing program.

In AX 3.8, Workbench supports rendering of SVG 1.1 images.

---




**NOTE:** Interactive features of SVG images, such as embedded Javascript-based animations, are not supported. For more details, see [About SVG rendering in Workbench, page 63](#)

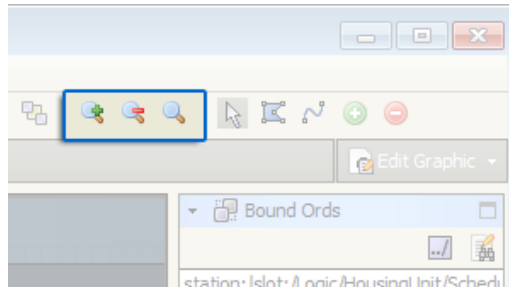
---

### About Zoom capability in Px Editor

In AX 3.8, you can zoom in and out on Px pages while in Px Editor Edit mode. The Zoom feature changes magnification of the Px page. Zoom-in increases magnification for a close-up view that is helpful for detailed work. Zoom-out reduces magnification of the page allowing you to see the whole page layout. Also, you can reset the zoom level back to 100%.

While editing a Px page, you will see three new icons in the upper-right area of the toolbar:

**Zoom In** () , **Zoom Out** () , and **Reset Zoom** () .

**Figure 23. Zoom icons**

These buttons all adjust the magnification level of the page. Zoom out to get a better view of the overall layout, and zoom in to achieve finer precision when positioning widgets.

The current zoom level is visible in the status bar at the lower right corner of the window. The zoom level can range from x0.2 (maximum zoom out) to x10.0 (maximum zoom in).

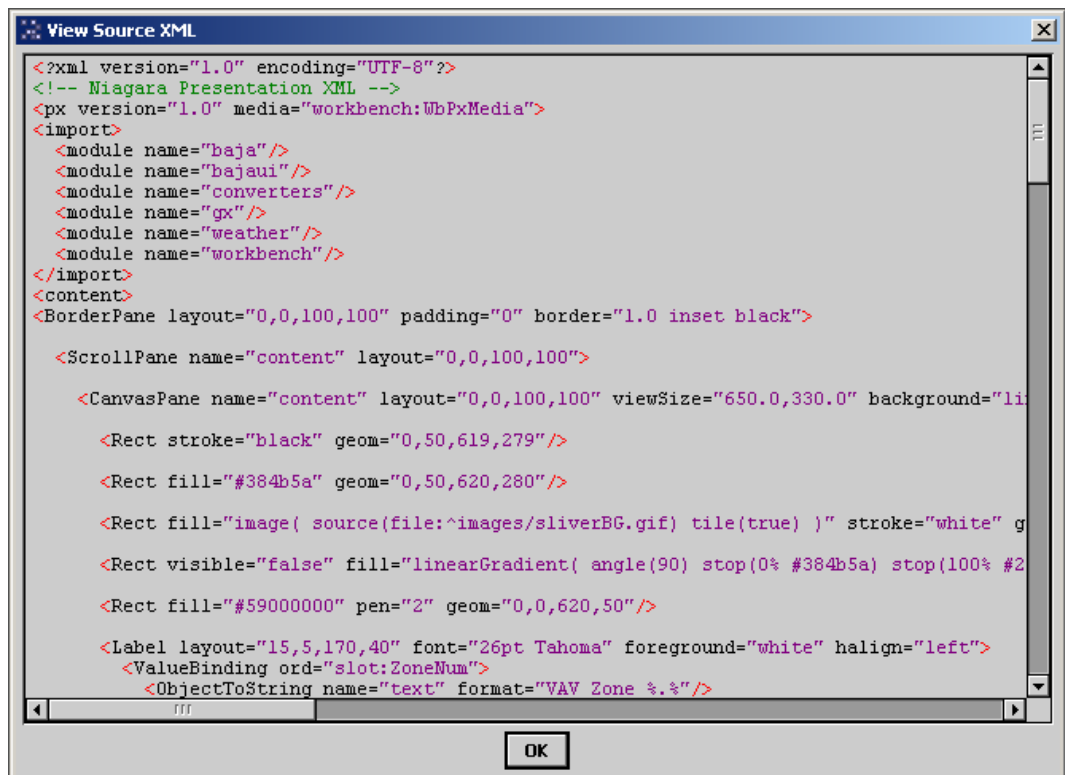
When navigating back and forth between Px pages, the zoom level will be remembered from page to page. To reset the zoom level to x1.0 (100%), just click the Reset Zoom button.

## About the View Source XML dialog box

The View Source XML dialog box allows you to view the source XML of a Px file in a separate read-only window.

Although you cannot edit the Px file in this window, you can copy and paste text from the window into the Text File Editor—or any other text editor, such as Notepad++.

The View Source dialog box, shown below, is a simple window with an OK button that closes the window when you click it.

**Figure 24. View Source XML dialog box**

## About Widgets

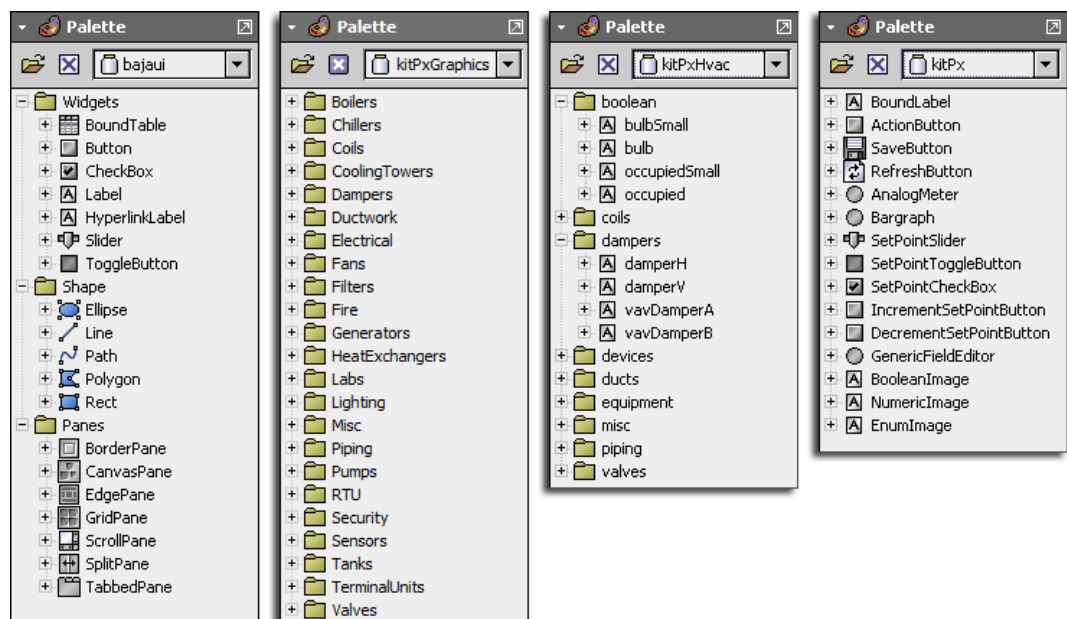
Widgets are components that provide visualization. Using Px Editor, you can work with widget properties to define user interface functions for control and information display .

User widgets can process input in the form of mouse, keyboard, and focus events. User events are grouped into these same event categories. The `bajau` module, the `kitPxGraphics` module, the `kitPxHvac` module, and the `kitPx` module all contain widgets that you can use for building rich user interfaces. You can find these widgets by opening them under the palette side bar, shown below, or you can find them using the Make Widget wizard.

In AX 3.8, a number of new widgets have been added to the `kitPxGraphics` module. The new widgets include:

- high-detail images of several different actuators in both large and small sizes
- static images of a JACE controller in three different sizes
- various animated widgets

**Figure 25. Widgets in the `bajau`, `kitPxGraphics`, `kitPxHvac`, and `kitPx` palettes**



The widget tree side bar provides a good illustration of the hierarchy structure of all widgets in a Px file (refer to “About the widget tree side bar” in the for a summary).

Some complicated widgets have mini-frameworks all to their own, such as the table widget, tree widget, treeTable widget, and textEditor widget. Refer to the “Baja Widget Toolkit” in the for more details about these types of widgets and for developer-level information about widgets.

The following sections discuss some of the primary characteristics of widgets.

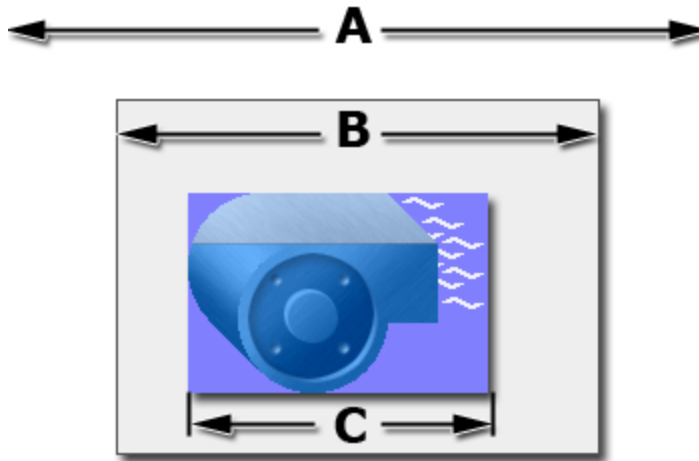
### Widget layout

A tree-type hierarchy of Px elements defines widget relationships and physical layout, in a structure where parent-child relationships exist between components.

All widgets occupy a rectangular area that is defined by a position and a size. The position of a widget is defined by its x, y coordinates relative to its parent’s coordinate system and each widget may have a default (or preferred) size. The size of the widget is defined by the width and height properties. The figure below shows a simple layout of a widget (C), held by a parent widget (B), that is the child object of a widget (A).

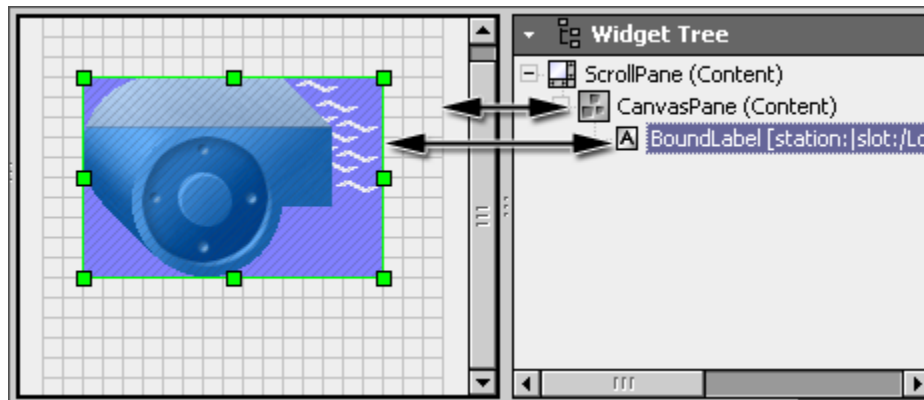
**NOTE:** If either of the child objects are positioned outside of the view area of the parent, they will be clipped and not be totally visible.

**Figure 26.** Scroll pane (A) holds canvas pane (B) with widget (C)



Some widgets are designed specifically to be container widgets that hold other widgets. These widgets are called *panes*. The figure below shows the pane hierarchy in the widget tree and on the Px editor canvas. Refer to [Types of panes, page 66](#) for a list of commonly used panes.

**Figure 27.** Pane hierarchy in the widget tree



### Types of panes

Summary of commonly used panes (widgets designed to be containers for child widgets).

Widgets that are designed to be containers for child widgets are called panes. Different types of panes provide different functions. The following list is a summary of some commonly used panes:

- **BasicMobilePane**  
the default pane for mobile Px pages and required in the root of a mobile Px file.
- **CanvasPane**

used for absolute positioning.

- **BorderPane**

used to wrap one widget and provide margin, border, and padding similar to the CSS box model.

- **EdgePane**

supports five potential children: top, bottom, left, right, and center. The top and bottom widgets fill the pane horizontally and use their preferred height. The left and right widgets use their preferred width, and occupy the vertical space between the top and bottom. The center widget gets all the remaining space. See the figure below for an example.

- **GridPane**

lays out its children as a series of columns and rows. You can configure extra space in the rows and columns a number of ways using the pane's properties.

- **MobileGridPane**

allows you to create row and column spacing on your mobile Px view.

- **SplitPane**

supports two children with a movable divider between them.

- **TabbedPane**

supports multiple children - only one is currently selected using a set of tabs.

---

**NOTE:** Even though only one tab is visible at a time, all tabs must be loaded on the page, even if they are not visible. If many tabs, with a lot of information, are on a single page, the page may load very slowly.

---

- **ScrollPane**

supports a single child that may have a preferred size larger than the available bounds. The scroll pane provides a set of scroll bars for viewing areas of the child widget that go outside of its bounds.

- **ReportPane**

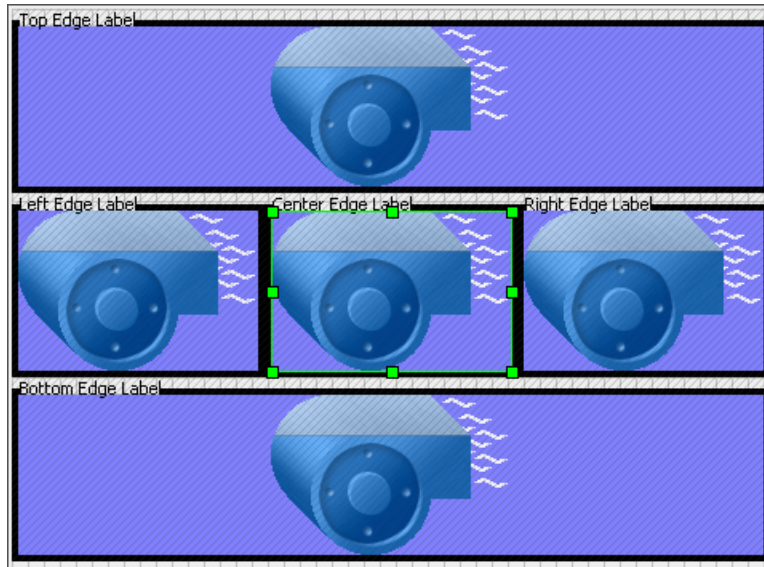
supports headers and footers in report PDFs. The ReportPane allows its content to span multiple pages, and allows each page to have a common logo as well as a page number and timestamp.

---

**NOTE:** Report panes should be at the root of a Px file. If you place a report pane inside a scroll pane, then page-spanning is disabled.

---

The figure below illustrates the edge pane layout using borders and labels.

**Figure 28. Edge panes, borders, and labels on a canvas pane****Widget painting**

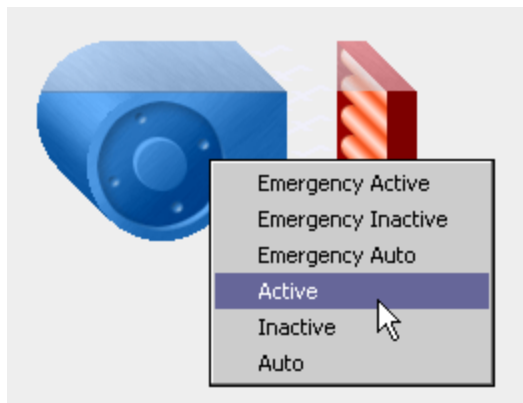
Painting defines how widgets present themselves graphically (using: colors, transparencies, and so on), as well as clipping. .

Graphics are always translated so that the origin 0,0 is positioned at the top, left corner of the widget. The graphic's clip (visible area) is set to the widget's size. Alpha and transparent pixels blend with the pixels already drawn. Widgets are drawn in property order, the first widget is drawn first (at the bottom), and the last widget drawn last (on the top). Effective z-order is the reverse of property order.

**Widget commands**

Widgets can have user commands that are commonly used with buttons and menu actions.

Toggle commands are commonly used with a checkbox, radio button, and other items. Menu commands are available and used on fans, coils, and other widgets, as shown in the figure below.

**Figure 29. Widget commands**

For more information about commands, refer to the “Baja Widget Toolkit” in the *NiagaraAX Developer Guide*.

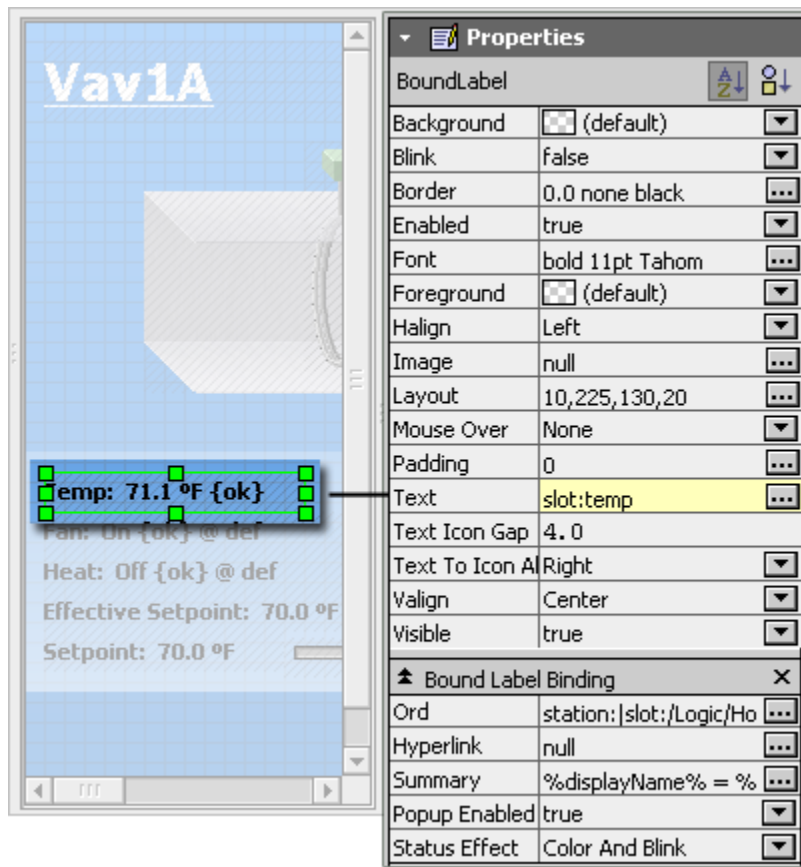
## Widget properties

Widget properties define many of the features, behaviors and appearance characteristics of widgets. By editing widget properties, you set or change the widget attributes.

In the Px Editor, you can edit widget properties using the properties side bar or properties dialog box and their secondary dialog boxes. Many of the properties (those that appear with the ellipsis icon) have these secondary dialog boxes that display when you click a property field. The secondary dialog box provides more detailed property fields and options.

The figure below shows the property side bar fields for a bound label. In this example, the text property of the bound label is “bound” to the “slot:temp” value, as shown in the text property field. The binding of this value allows the dynamic presentation of the temperature information in the Px view.

**Figure 30. Widget properties**



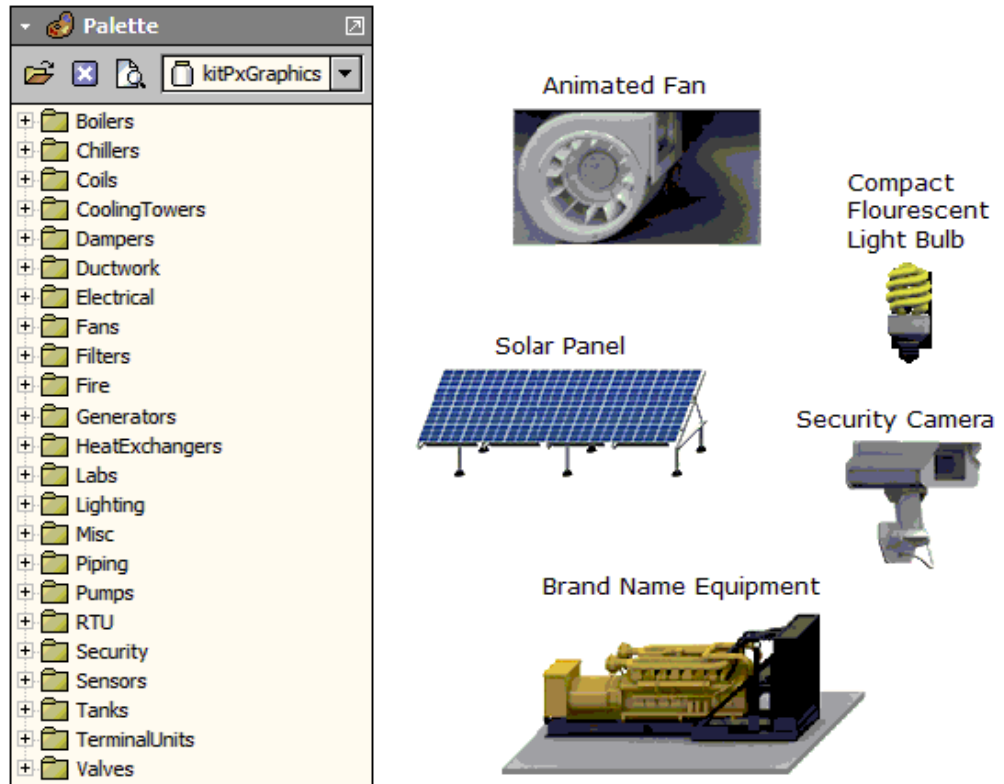
## About the kitPxGraphics palette

In AX-3.7 and later, the Niagara installation includes a new graphics library module, kitPxGraphics, that contains new images and updated versions of many HVAC images found in the kitPxHvac palette, as well as many new images.

The images in kitPxGraphics are professionally designed, higher quality images than those in kitPxHvac. Consequently, the files take up more storage space (for example, the file size for the animated fan image increased by 40%). The figure below shows the kitPxGraphics palette along with sample images.

**NOTE:** The new kitPxGraphics palette is NOT a replacement for kitPxHvac. The kitPxHvac palette will continue to be available in AX-3.7 and later versions.

**Figure 31. kitPxGraphics palette and sample images**



Due to the larger file sizes and a greater number of images the kitPxGraphics module is quite large (19MB) compared to other graphic library modules. At that size, most JACEs in the field are not able to store the full kitPxGraphics module. That, coupled with the fact that most job sites will require only a fraction of the images in kitPxGraphics, has led to a change in how PxEditor handles image copying. The new functionality enables PxEditor to conserve storage space by copying only individual images to a remote station rather than copying the entire graphics module, as was done in earlier versions of AX .

**NOTE:** This applies only to images contained in modules that are larger than 2MB, such as kitPxGraphics.

When using PxEditor to copy either an image from a module or to copy a label from a module's palette that contains images, only the referenced image is automatically copied to the station and the ord for the label is changed to reflect the new location of the image: ^px/module-Name/pathToImage.

**NOTE:** In order for PxEditor to copy an image file to a remote station, the station must be running.

The following rules apply when PxEditor copies widgets from the kitPxGraphics palette:

- If the module already exists on the remote station, PxEditor does not need to copy the image, it is already in the module on the station.



- If the module does not exist on the remote station, PxEditor copies only the referenced individual image to: `^px/moduleName/pathToImage`.
- If the station is running locally, PxEditor copies only the referenced individual image (as in step #2) if the module size is greater than 2MB.

---

**NOTE:** This size is set via `niagara.ui.px.maxImageModuleFileSize` in `!lib\system.properties`.

---

- If the station is offline (not a running station) no image copying occurs since PxEditor is not able to copy files to an offline station.

### ***About third-party graphics collections***

You can use images from a third-party graphics collection in Workbench. You must manually copy the needed images onto a JACE and then add the bound labels to a Px page and create the graphics bindings.

An alternative to buying a 3rd-party graphics collection is to create your own graphics library module and `module.palette` file of bound labels for the graphics. The `module.palette` file makes the graphics and label bindings visible in PxEditor's **Make Widget** dialog box. Also, putting the graphics files in a module allows you to take advantage of the new individual image copying functionality in PxEditor. Refer to the “*Build*” section of the for detailed information on building modules.

## **About developing for portability**

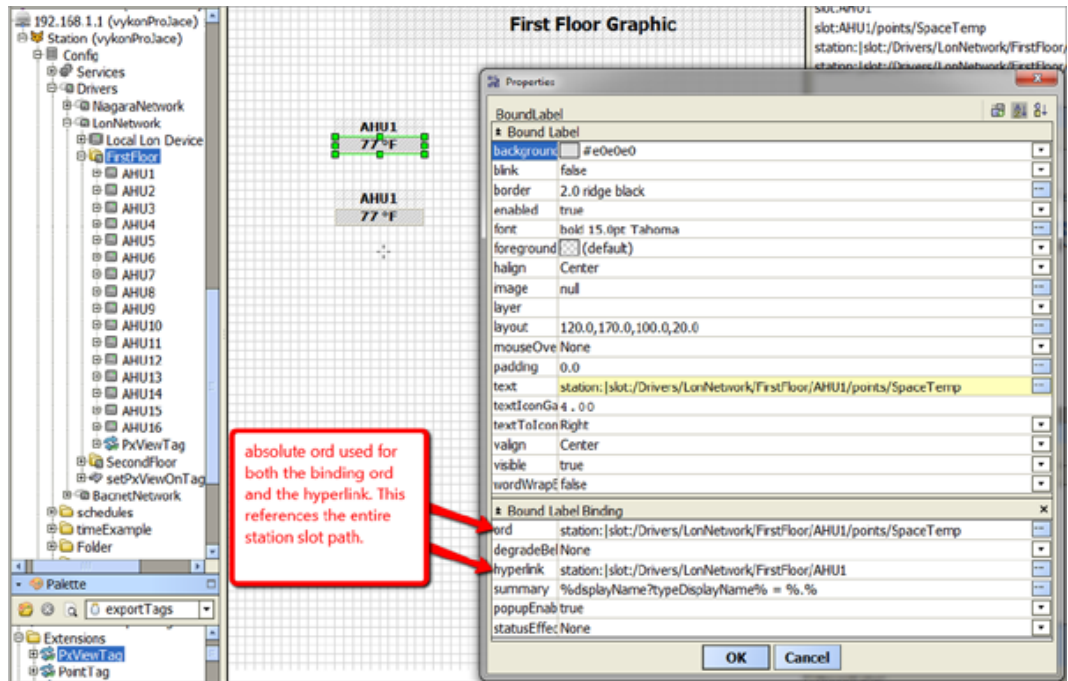
Develop Px Views that you can easily use in multiple scenarios by specifying relative ORDs in bindings and hyperlinks.

For purposes of this discussion, the term “portability” means reusability. In other words, being able to use the same Px View in different JACE stations without having to edit the ORD properties for bindings and hyperlinks. Another example of portability, is being able to use the same Px view in a JACE station and a Supervisor station without having to edit ORD properties. You can develop portable Px views by avoiding the use of *absolute* ORDs in bindings and hyperlinks. The reason to avoid using absolute ORD properties in Px views is that they are not portable between stations.

### **Bound label using absolute ORD**

The floor plan graphic shown below has a bound label configured with a hyperlink to a detailed graphic for a specific device. You can see that both the ord binding and hyperlink properties use an absolute ord, meaning the value specifies the entire station slot path.

**Figure 32. Floor plan graphic using components configured with absolute ORDs**



If you wanted to use the same floor plan graphic in the Supervisor station it you would have to create Niagara network proxy points and assign the same Px view (or you could use Px view export tags). In either case, the absolute slot path used in the Supervisor must be different than that used in the JACE.

- **Supervisor absolute ORD**

```
station:|slot:/Drivers/NiagaraNetwork/VA/Richmond/vikonProJace/points/FirstFloor/AHU1
```

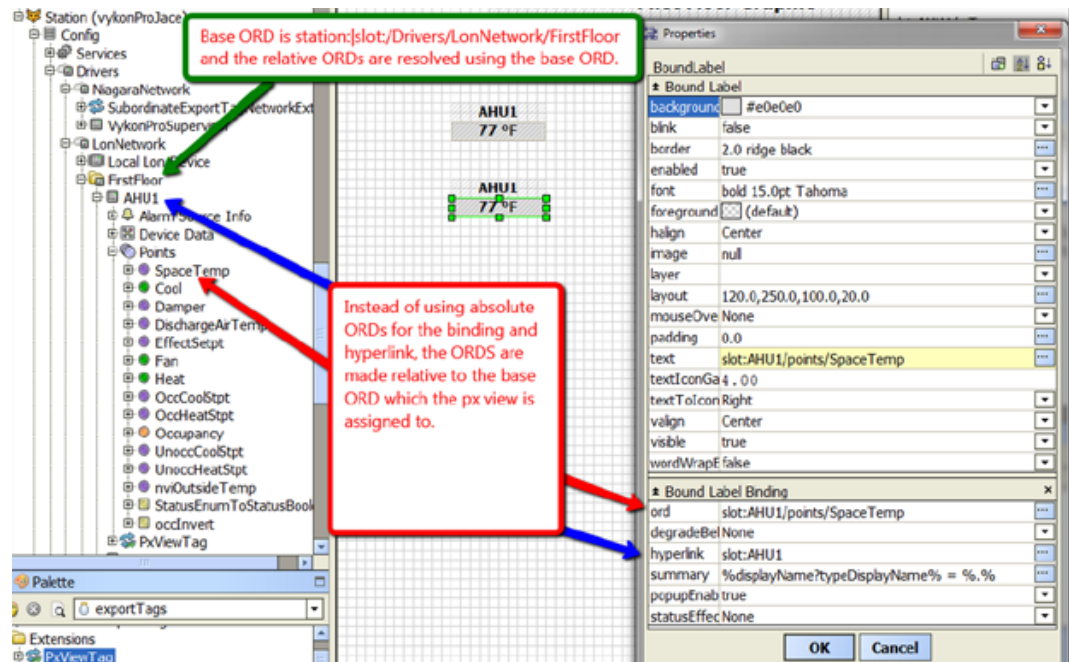
- **JACE absolute ORD**

```
station:|slot:/Drivers/LonNetwork/FirstFloor/points/FirstFloor/SpaceTemp
```

### Bound label using relative ORD

Here, you have the same scenario but in this case using *relative* ORDs instead of absolute ORDs.

**Figure 33. Floor plan graphic using components configured with relative ORDs**

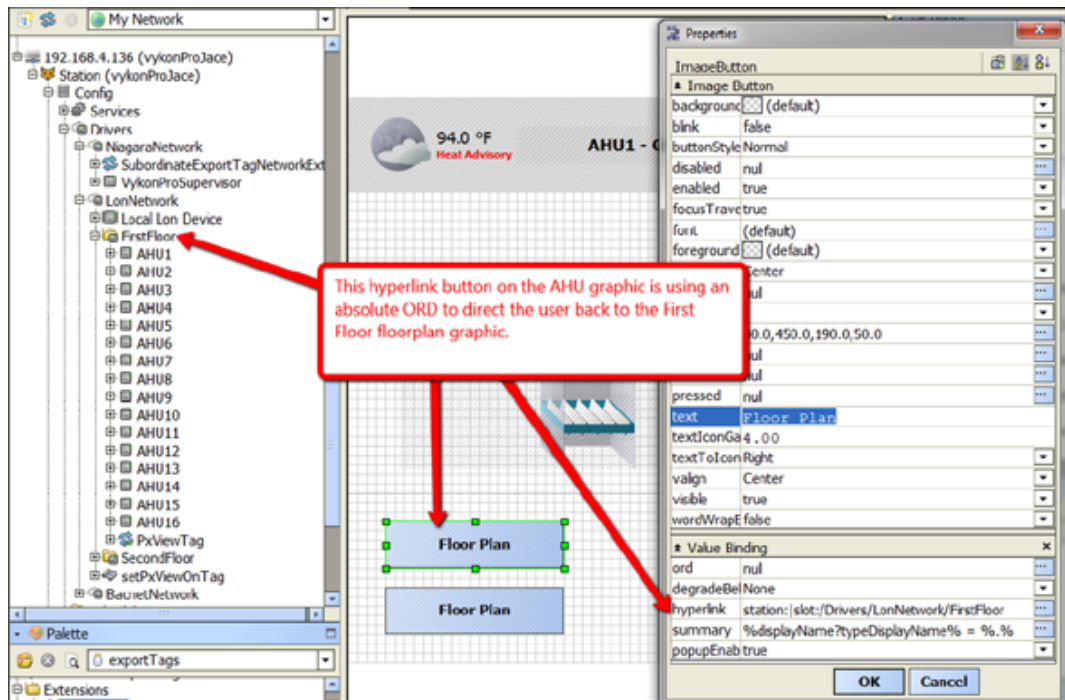


The same relative ORDs can be used in both the JACE and the Supervisor station since it is being applied against different base ORDs

### Hyperlink using absolute ORD

The following example, the user has hyperlinked from the floor plan graphic to a more detailed graphic of the air handling unit (AHU). This detailed AHU graphic includes a button that is a hyperlink back to the floor plan graphic (FirstFloor). The button hyperlink is configured with an absolute ORD.

**Figure 34.** Graphic that includes a button with a hyperlink configured with an absolute ORD



In order to use the above graphic in the Supervisor station, you must use Niagara network proxy points and assign a Px view or by using a Px view export tag.

The absolute ORD to the First Floor graphic in the JACE is different from that in the Supervisor.

- **Absolute ORD in JACE**

```
station:|slot:/Drivers/LonNetwork/FirstFloor
```

- **Absolute ORD in Supervisor**

```
station:|slot:/Drivers/NiagaraNetwork/VA/Richmond/vykonProJACE/points/FirstFloor
```

### Hyperlink using relative ORD

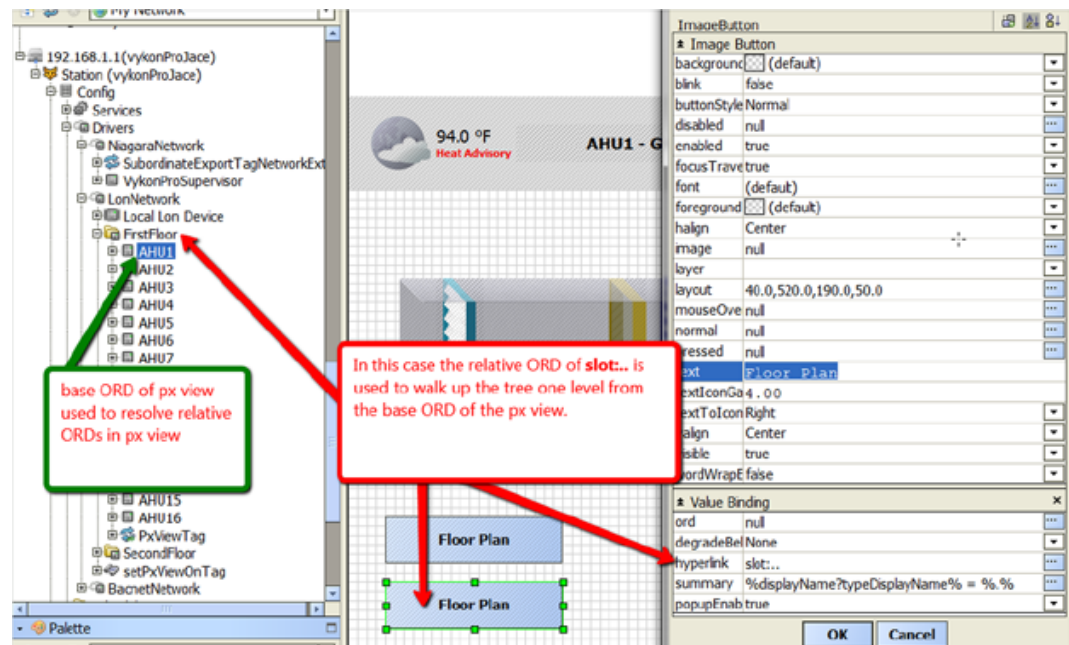
Using relative ORD properties, it is possible to navigate back up the tree by entering two periods in the path, similar to how directories can be navigated in a DOS command window.

---

**NOTE:** You can back up multiple levels using this syntax: “. . .”

---

**Figure 35. Graphic includes a button with a hyperlink configured with a relative ORD**



- **Relative ORD in Px page**

slot:..

- **Base ORD in JACE**

station:|slot:/Drivers/LonNetwork/FirstFloor/AHU1

- **Base ORD in Supervisor**

station:|slot:/Drivers/NiagaraNetwork/VA/Richmond/vykonProJACE/  
points/FirstFloor/AHU1

## Absolute versus Relative ORDs in HTTP/Fox tunneling

When using HTTP/Fox tunneling most people use absolute ORDs for hyperlinks such as below. Where the red text is the reference to the Supervisor, the green text is the tunnel reference to the JACE station, and the purple text is the ORD in the JACE station.

**Figure 36. HTTP/Fox tunneling using absolute ORDs for hyperlinks**

ip:192.168.4.29|fox:/192.168.4.136|station:|slot:/home

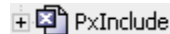
Assuming that this is a Px graphic in the Supervisor station, the base ORD of the Px view will be the Supervisor station and can be used to make a relative tunnel hyperlink to the JACE.

Once you are tunneled to the Px view in the JACE, the base ORD of the Px view includes the information about connecting to the Supervisor.  
ip:192.168.4.29|fox:/192.168.4.136|station:|slot:/home|view:home

Using the syntax of proxyHost:| will essentially strip off the remaining tunnel ord syntax (fox:|192.168.4.136:|) and return the user to the Supervisor station. This could be useful when trying to access the system graphics from both the LAN and WAN. Meaning, the Supervisor station would have a different IP address depending on whether you are accessing while connected to the LAN or connecting via the WAN or public IP via the internet.

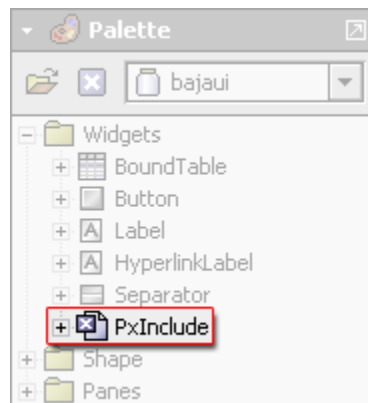
## About the PxInclude Widget

This widget lets you embed a single Px file in another Px file.



The PxInclude is a widget located in the NiagaraAX bajai palette, as shown in the following illustration.

**Figure 37. The bajai Palette contains the PxInclude widget**



This widget provides a way for users to embed a single Px file in another Px file. The key benefit of this feature is the ability to create reusable Px views and embed them into one or many other Px views. The following sections describe the widget:

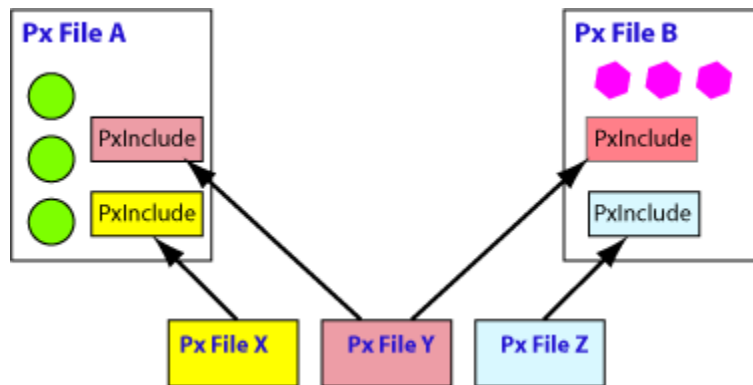
- [PxInclude Widget concepts, page 76](#)
- [Types of PxInclude Widget properties, page 78](#)
- [Use PxInclude Widgets , page 79](#) (including [About ORD Variables, page 80](#), [Workflow for using the PxInclude Widget and ORD Variables, page 37](#), and [Use PxInclude Widgets , page 79](#))

### ***PxInclude Widget concepts***

The PxInclude widget may have a single Px file associated with it, providing one "include" per widget, however, you can put more than one PxInclude widget in a Px file.

Depending on the design of a Px file, a single "parent" Px file may contain many "child" Px files by using many PxInclude widgets.

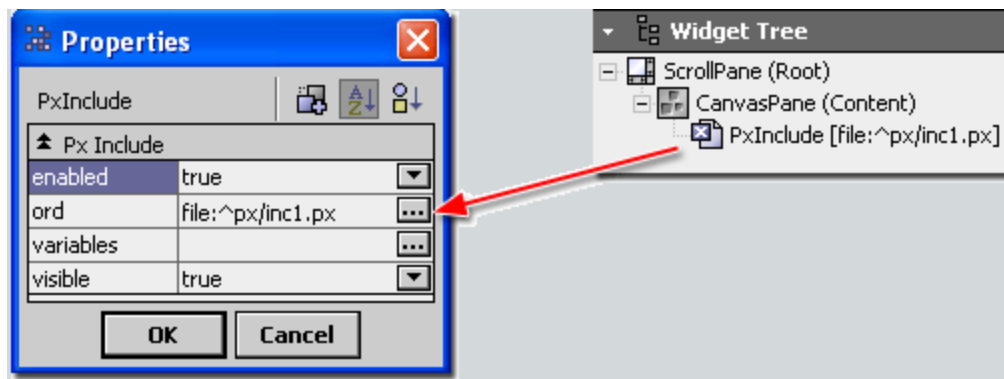
The following diagram illustrates the basic concept of embedding one Px file in one or more other Px files.

**Figure 38. PxInclude Concept Diagram**

Note the following about the Px files represented in this diagram:

- Px files X, Y, and Z are included in other Px files
- Px files X, Y, and Z may display as independent views, as well as be embedded using the PxInclude widget.
- Px file Y is included in both Px file A and Px file B.
- Changes to files X, Y, or Z are displayed in their parent Px files (A or B), via the PxInclude, when the parent view is refreshed.

PxIncludes work by defining an ord property value that points to the "included" file, as shown in the following illustration.

**Figure 39. PxInclude points to the file to be embedded**

This example shows the "inc1.px" file embedded in another Px file. The embedded filename is visible in both the **Properties** dialog box and in the Widget Tree.

When PxIncludes are added to a Px file, the "include" markup appears as follows in the Px code (visible in the "Text File Editor" view).

**Figure 40. Simple PxInclude markup in a parent Px file**

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Niagara Presentation XML -->
<px version="1.0" media="workbench:WbPxMedia">
<import>
  <module name="baja" />
  <module name="bajau" />
  <module name="gx" />
  <module name="history" />
  <module name="workbench" />
</import>
<content>
<ScrollPane>

  <CanvasPane name="content" viewSize="500.0,400.0">
    <PxInclude layout="100,80,350,250" ord="file:^px/incl.px" />
  </CanvasPane>
</ScrollPane>
</content>
</px>

```


**Types of PxInclude Widget properties**

The PxInclude Widget has the following properties:

- **enabled**

This property has two options: true and false. Setting the option to false causes the widget to stop working. Values may display but are not reliable and do not update.
- **ord**

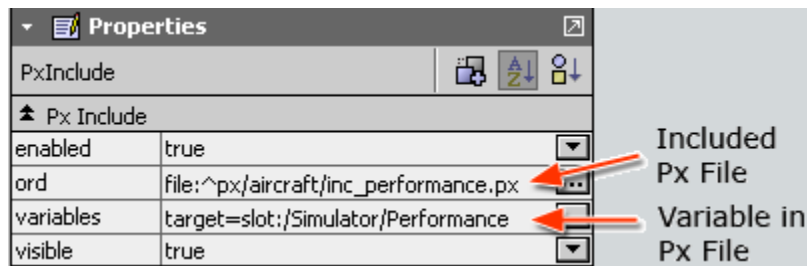
The value of this property (an ORD) identifies the Px file that is linked using the PxInclude widget. See an example in the figure below.
- **variables**

This optional property identifies the value of any variables in the Px file specified by the ord property. Clicking on the variable field displays the Variables dialog box, where you can define the variable value. See an example in the figure below. Also see [About ORD Variables, page 80](#).
- **layout**

This property contains parameter fields for values that prescribe PxInclude widget size and location.
- **visible**

This property has two options: true and false. Setting the option to false causes the widget to not display.



**Figure 41. Example PxInclude Properties Palette in the Px Editor**

Note the following about this illustration:

- The **ord** property `file:^px/aircraft/inc_performance.px` specifies to include the Px file named `inc_performance.px`. This file is located in the station's (^) `px/aircraft` folder.
  - The **variables** property value indicates that the included file has a single variable named "target" and that this variable is defined as `slot:/Simulator/Performance`.
- The **visible** property value is set to the default value `true`.

### Use PxInclude Widgets

Use the Make Widget Wizard to select the desired Px file and choose which variables to bind.

Like other widgets, you can add a PxInclude to the Px Editor canvas in several ways, including the following:

- **Drag (cut and paste) the widget from a palette**

This method simply drops a widget onto the Px Editor canvas, adding the widget to the current Px file. After adding the widget to the Px canvas, you must edit the `ord` property to add the desired Px file.

- **Drag (cut and paste) a Px file from the workbench Nav tree**

This method adds a PxInclude widget to the current Px file with a Px file already connected to the PxInclude `ord` property.

- **Drag (cut and paste) a component from the Nav tree**

This initiates the **Make Widget Wizard**. This method allows you to use the Make Widget Wizard to select the desired Px file and choose which variables to bind.

---

**NOTE:** Only the `ord` variables that are selected in the bottom left window of the Make Widget Wizard are available for configuring after the PxInclude file is imported. See [About ORD Variables, page 80](#).

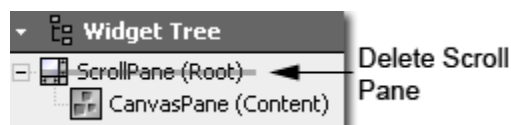
---

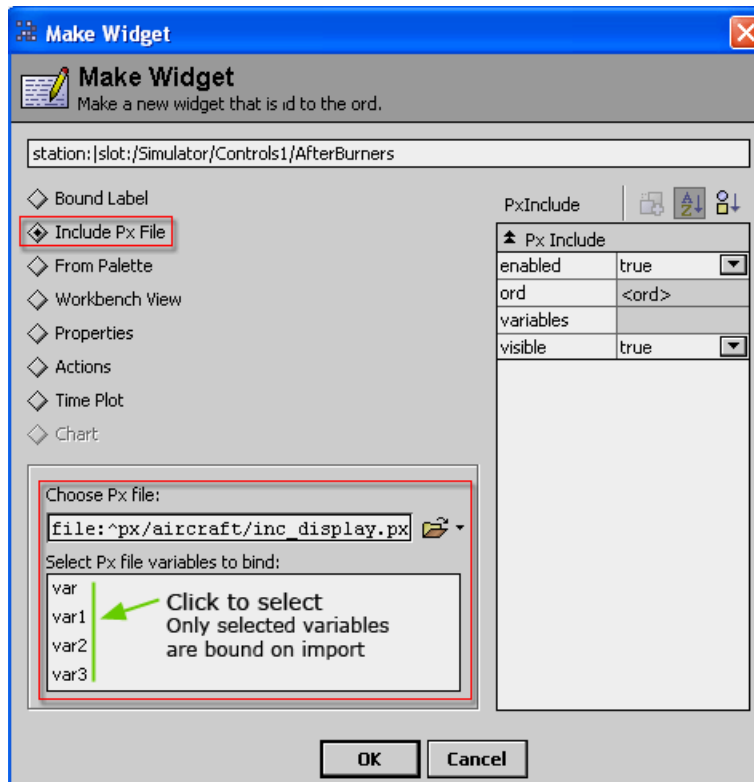


---

**NOTE:** For the Px file that you are including, you probably want to remove the root scroll pane that comes with the default Px file. Just use a canvas pane as the root. The scroll pane has a property called `border policy`, which is set to 'always' by default. This border is visible when the include is viewed in its parent Px file.

---

**Figure 42. Remove the scroll pane from a Px Include to avoid displaying a border**

**Figure 43. Adding a PxInclude Using the Make Widget Wizard**

When you complete the Make Widget Wizard, you may have the PxInclude widget configured fully, or edit it further, if needed.

### About ORD Variables

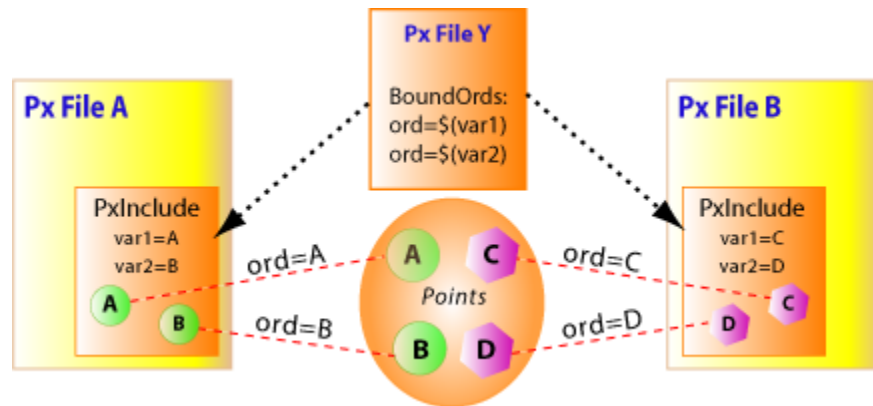
ORD variables are optional portions of an "ORD binding" that you can create in order to provide more flexibility or "reusability" in your Px files.

You can use a variable for the complete ORD or identify a variable part of an ORD by using the following special syntax in a bound component ord property field.  $\$(var)$

Where:

- $\$$  is a special character used to identify an ORD variable
- $(var)$  is a named variable (may be any text string) inside a set of parentheses. The name in parentheses is the name that is substituted by a literal text string in the ord Variables dialog box after the child Px file is included in the parent Px file.

The following illustration shows the general relationships between Px files that use ORD variables in PxInclude widgets.

**Figure 44. PxIncludes Allow Px File Reuse With Variable ORDs**

Note the following about this example:

- Px File Y uses two ORD variables:  $\$(var1)$  and  $\$(var2)$ .
- Px File Y is embedded in two different Px files using PxInclude widgets.
- In Px File A, var1 is defined as "A", resulting in a binding "ord=A". The "ord=A" value binds the widget to a "Point A", so a value from that "Point A" is displayed in the parent view (Px File A).
- In Px File B, var1 is defined as "C", resulting in a binding "ord=C". The "ord=C" value binds the widget to a "Point C", so a value from that "Point C" is displayed in the parent view (Px File B).
- Similar to the descriptions above, in Px files A and B, var2 is defined as ord=B and ord=D, respectively, resulting in different point values displayed in Px Files A and B.

## About Animating Graphics

Animated graphics are comprised of one or more widgets assembled in a Px file, available for display using the Workbench display media types. Animated graphics change, or “update”, based on data values that come from object sources that are connected (or “bound”) to them.

A graphic can be as simple as a single word of text “ON” or a number “72”, or it can be an animated image (rotating fan). Widgets provide the graphic visualization of data in Workbench, so animated graphics are comprised of one or more widgets assembled in a Px file, available for display using the Workbench display media types. Widgets are animated by binding any widget properties to a legitimate data source. This means that you can connect numeric values to widget properties that use numeric values and you can connect binary values to objects that can use binary values. By animating the properties of a widget, you can control text and image appearance as well as a change a widget’s location on the page and even its visibility.

The following sections describe the different concepts that relate to animating graphics.

- [Data binding, page 81](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)
- [About relative and absolute bindings, page 91](#)

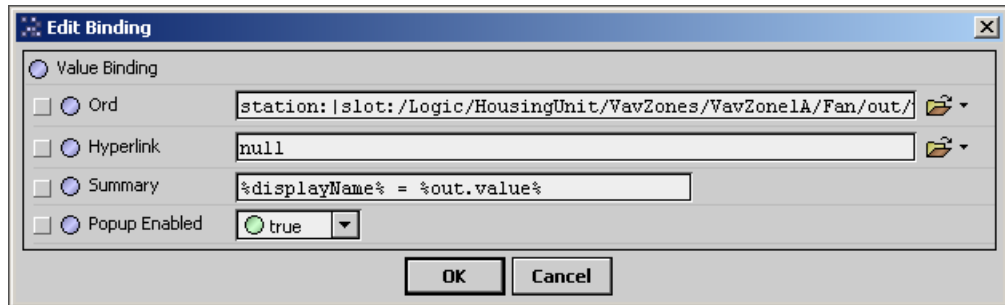
### Data binding

Bindings are established between a widget and an object. Binding provides real-time information for presentation.

All widgets may be bound to data sources using data binding. Bindings are established between a widget and an object using an ORD. A single binding consists of a single widget-object relationship. A binding's ORD property identifies the location of the object that updates and animates the widget.

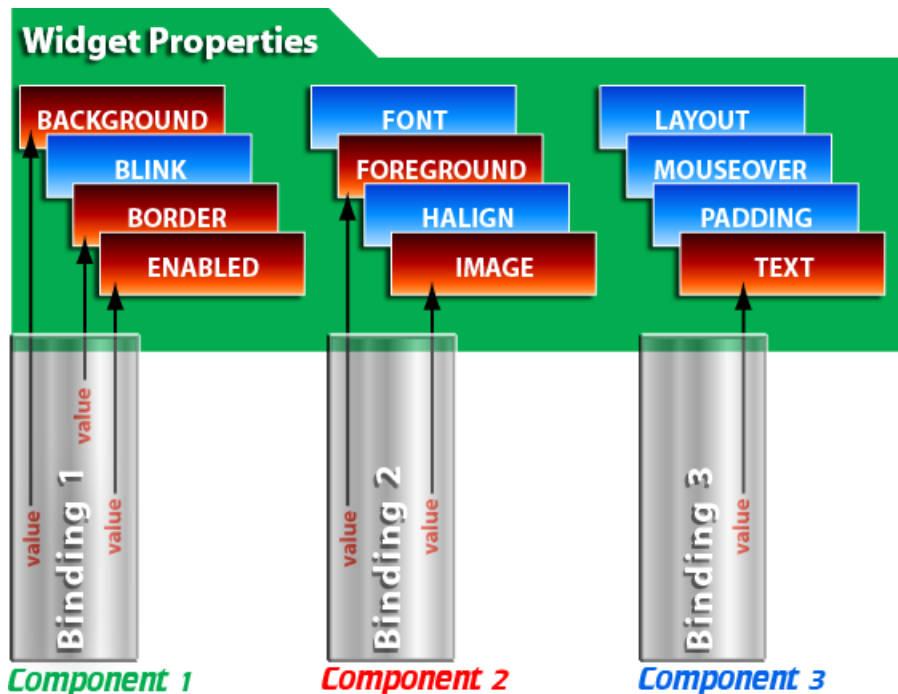
For example, the most common type of binding, the value binding, provides some of the typical functions that are associated with building real-time information for presentation as both text and graphics. This includes support for mouse-over status and right-click actions. Additionally it provides a mechanism to animate any property of its parent widget using converters that convert the target object into property values. [Figure 45. Widget data binding, page 82](#) shows a value binding to a fan widget as viewed in the **Edit Binding** dialog box.

**Figure 45. Widget data binding**



[Figure 46. Widget with three bindings, page 82](#) illustrates the object-to-widget property binding concept. In this example, a widget has three separate data bindings. This means that each binding is coming from a different object and therefore each binding has a different ORD that defines its binding. Each binding provides access to an object's values so that they may be used, as required, to animate the widget properties.

**Figure 46. Widget with three bindings**

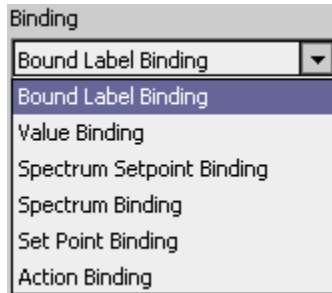


## Types of data bindings

There are different types of bindings that may be used with widgets.

Some bindings work with only a certain type of widget (for example, a bound label binding) and other binding types may be used with several types of widgets (for example, a value binding). [Figure 47. Types of bindings, page 83](#) shows bindings as they appear in a Workbench options list.

**Figure 47. Types of bindings**



Following, is a list and short description of each binding type.

- Action binding
  - invokes an action on the binding target component when an event is fired by the parent widget. See [About action bindings, page 90](#) for details.
- Bound Label binding
  - connects a value to a bound label widget. See [About bound label bindings, page 86](#) for details.
- Field Editor binding
  - used to bind field editor components to an object. See [About field editor bindings, page 90](#) for details.
- Popup binding
  - used to display a Px view in an additional “popup” window that you can specify and configure. See [About Popup Bindings, page 91](#) for details.
- Setpoint binding
  - used to display the current value of a "setpoint" and also to provide the ability to modify it. See [About spectrum setpoint bindings, page 89](#) for details.
- Spectrum binding
  - used to animate a widget's brush (color) property. See [About spectrum bindings, page 88](#) for details.
- Spectrum setpoint binding
  - used in conjunction with a spectrum binding. See [About spectrum setpoint bindings, page 89](#) for details.
- Table binding
  - used to bind table data in a bound table. See [About table bindings, page 90](#) for details.
- Value binding
  - used to bind to values that are typically under a component. See [About value bindings, page 87](#) for details.

## Types of binding properties

One or more of the following properties may be included with various binding types.

- **actionArgument**

This property is used with certain widgetEvents to specify the action to take when the widgetEvent is triggered. For example, a mouseEvent (such as a “click”) on a widget can change a boolean status from “true” to “false” or prompt the user to select a setting. Argument options, like the widgetEvents, themselves, are context sensitive, depending on widget and binding type.

- **degradeBehavior**

This option allows you to specify how the object behaves when binding communications are not available. If a binding is not usable, then this property allows the designer to choose how the UI is gracefully degraded. For example, if the user doesn't have permission to invoke a specific action, then a button that is bound to that action can be dimmed or hidden entirely. To preserve backward compatibility, the default degradeBehavior is none.

- **extent**

This spectrum binding property represents the total range of the bound value which maps from low to high.

- **highColor**

This spectrum binding property specifies the color that is used for the highest value assignment for a spectrum color binding. The high color displays when the bound target is greater than  $\text{setpoint} + \text{extent} / 2$ . As the bound value decreases below the maximum value specified, the color approaches the color set by the Mid Color property.

- **hyperlink**

This property provides a link to another object. When the field is completed, the hyperlink is active in the browser or in the Px viewer.

- **icon**

This property value specifies an icon, if desired, to appear in the top left corner of a popup window. The icon property is not available with the popup binding.

- **increment**

This property is a value that can be set to increment (increase) or decrement (decrease) a current value by the assigned amount. A positive number in this property field increments a value; a negative number in this field decrements a value.

- **lowColor**

This spectrum binding property specifies the color that is used for the lowest value assignment for a spectrum color binding. The low color displays when the bound target value is less than  $\text{setpoint} - \text{extent} / 2$ . As the value bound to this property increases above the minimum value specified, the color approaches the color set by the Mid Color property.

- **midColor**

This spectrum binding property specifies the color that is used for the middle value assignment for a spectrum color binding. This color is displayed when the bound value is exactly at the setpoint. As the binding value increases above this point, the displayed color approaches the color set by the High Color property. As the bound value decreases below this point, the displayed color approaches the color set by the Low Color property.

- **modal**

This property applies to Popup Bindings (see [About Popup Bindings, page 91](#)). When the `modal` property is set to true, the popup window associated with the Popup Binding is a modal window.

---

**NOTE:** A modal window is a child (or secondary) window that appears in front of a parent window and typically does not allow interaction with the parent window. The modal window created using this property is always on top but does not prevent a user from interacting with the parent window.

---

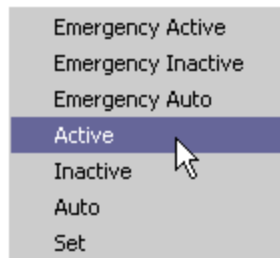
- **ord**

This property specifies the ORD that is bound to the widget. This field must have a value if you want the widget to be bound. In a Popup binding this is the path that designates the component view that is displayed in the popup window.

- **popupEnabled**

This property has `true` or `false` options and allows you to specify if you want to allow a secondary popup (right-click) menu (see [Figure 48. Example popup \(right-click\) menu, page 85](#)) to appear when a user clicks on this label from a browser or Px viewer.

**Figure 48. Example popup (right-click) menu**



If the option is set to `false` then the popup action menu is not available. You may use this if you want to prevent actions from being executed at a control point from a specific Px graphic while still allowing access to the action from a different Px graphic or property sheet.

---

**NOTE:** Other ways to make actions unavailable include:

- Hiding the action slot on the component makes the action unavailable but it would apply to every point of access in the station.
  - Actions can also be assigned operator or admin permission requirements to limit the access.
- 

- **position**

This property is used to designate an initial screen position for a popup window.

- **setpoint**

This binding property specifies the value that is used for displaying the Mid Color. For example, if you set this value to “70”, then the color that you define for Mid Color will be displayed when the bound value is “70”.

- **size**

This property is used to designate an initial screen size for a popup window.

- **statusEffect**

this property provides the following three options for enabling or disabling status effects.

- **Color**  
choose this option to enable a background color change when the bound value status changes.
- **Color and blink**  
choose this option to enable a background color change and a blinking effect when the bound value status changes.
- **None**  
choose this option to disable any effects based on bound value status changes.
- **summary**  
This property allows you to specify a display name for the widget in text or by means of a script.
- **title**  
This property value specifies the text that appears in a popup window title bar.
- **widgetEvent**  
This property allows you to specify one of several possible events: actionPerformed, focusEvent, invokeAction, keyEvent, mouseEvent. Possible actions are dependent on the type of widget and action. Some widgetEvents allow you to specify an actionArgument, as described, above.
- **widgetProperty**  
This property allows you to specify the property that you are targeting in the binding's parent widget. For example, you can use a spectrum binding that has a bound label parent to change the background property of that parent label by selecting "background" in the Widget Property options field. Or, you can change the foreground color by choosing "foreground". You can target only one value per binding but you may add additional bindings if you want to target more than one property in the parent widget.

### **About bound label bindings**

Connect a value to a bound label widget.

Bound label bindings are used exclusively for connecting a value to a bound label widget. Bound labels, which are located in the kitPx module, have properties that you can edit and access from the Px Editor properties side bar. The properties pane for the bound label binding is shown below.

**Figure 49. Bound label binding properties**

Bound Label Binding		X
Ord	station:  slot:;/Logic/HousingUnit/VavZones/	...
Hyperlink	null	...
Summary	%displayName% = %.%	...
Popup Enabled	true	▼
Status Effect	None	▼

Bound label binding properties include the following:

- ORD
- Hyperlink
- Summary
- Popup Enabled



- Status Effect

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

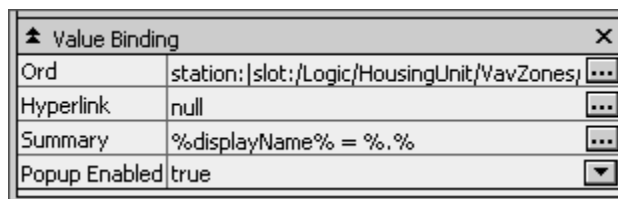
### About value bindings

You can use value bindings to support real-time graphics, mouse-over effects, and right-click actions.

Value bindings are used to bind to values that are typically under a component. Widget properties may be overridden by creating dynamic slots of the same name with a converter that maps the bound target object into a property value. This capability is what allows animation of any widget property. Value bindings support features such as real-time graphics, mouse-over, and right-click actions.

The properties pane for the bound label binding is shown in [About value bindings, page 87](#).

**Figure 50. Value binding properties**



Value binding properties include the following:

- ORD
- Hyperlink
- Summary
- Popup Enabled

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

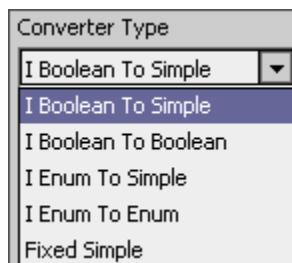
Refer to [Types of Converters, page 87](#) for a list of converter types.

### Types of Converters

Converters are used to change data from one type to another so that the data can be used to communicate with a particular widget property value.

. In most cases, when you animate a property, the correct data converter appears, by default, at the top of the list, as shown in [Types of Converters, page 87](#).

**Figure 51. Types of converters**



The following types of converters are available when using a value binding:

- Fixed Simple
- I Numeric to I Simple




- I Status to Simple
- Object to String

### About spectrum bindings

Use this binding to animate a widget's brush (color) property by mapping a numeric value into a color range defined by lowColor, midColor, and highColor properties.

The properties pane for the spectrum binding properties is shown below.

**Figure 52. Spectrum binding properties**

Spectrum Binding	
Ord	slot:/Logic/HousingUnit/VavZones/VavZone ...
Widget Property	background
Low Color	 ...
Mid Color	 ...
High Color	 ...
Setpoint	49.0
Extent	100.0

Spectrum properties include the following:

- ORD
- Widget Property
- Low Color
- Mid Color
- High Color
- Set Point
- Extent

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### About setpoint bindings

Use this binding to display the current value of a “setpoint” and also to provide the ability to modify it.

A setpoint is typically a status value property such as `fallback`. The setpoint binding ORD must resolve down to the specific property that is being manipulated. If it is bound to a component or to a read-only property, then the binding attempts to use a “set” action to save.

The properties pane for the spectrum setpoint binding is shown in [Figure 53. Setpoint binding properties, page 88](#).

**Figure 53. Setpoint binding properties**

Set Point Binding	
Ord	station:/slot:/Logic/HousingUnit/AirHandler/SetpointTei ...
Hyperlink	null
Summary	%displayName% = %.%
Popup Enabled	true
Widget Event	actionPerformed
Widget Property	value

Setpoint properties include the following:

- ORD
- Hyperlink
- Summary
- Popup Enabled
- Widget Event
- Widget Property

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### **About increment setpoint bindings**

Use this type of setpoint binding to increment or decrement a numeric value.

The properties pane for the increment setpoint binding is shown below.

**Figure 54. Increment setpoint binding properties**

Increment Set Point Binding		X
Ord	null	...
Hyperlink	null	...
Summary	%displayName% = %.%	...
Popup Enabled	true	▼
Widget Event	actionPerformed	▼
Increment	1.0	

Spectrum setpoint properties include the following:

- ORD
- Hyperlink
- Summary
- Popup Enabled
- Widget
- Increment

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### **About spectrum setpoint bindings**

This binding is used in conjunction with a spectrum binding to animate the midColor properties.

The properties pane for the spectrum setpoint binding is shown in [Figure 55. Spectrum setpoint binding properties, page 89](#).

**Figure 55. Spectrum setpoint binding properties**

Spectrum Setpoint Binding		X
Ord	station: slot:/Logic/HousingUnit/WavZones	...
Hyperlink	null	...
Summary	%displayName% = %.%	...
Popup Enabled	true	▼

Spectrum setpoint properties include the following:

- ORD
- Hyperlink
- Summary
- Popup Enabled

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### **About action bindings**

This type of binding invokes an action on the binding target component when an event is fired by the parent widget.

The ORD of an action binding must resolve down to a specific action within a component. Examples of actions include: “active”, “inactive”, “override”, and other commands. The properties pane for the action binding is shown in [Figure 56. Action binding properties, page 90](#).

**Figure 56. Action binding properties**

Action Binding	
Ord	station:/slot:/Logic/HousingUnit/Air
Widget Event	actionPerformed
Action Arg	

Action binding properties include the following:

- ORD
- Widget Event
- Action Arg

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### **About table bindings**

This type of binding is used to bind table data in a bound table.

Table bindings can bind to any “collection” using a BQL, SQL, or History ORD binding. For example, a table binding ORD might look like the following: `station:"slot:/bql:select toPathString,toString from control:BooleanPoint`

The properties pane for the table binding is shown in [Figure 57. Table binding properties, page 90](#).

**Figure 57. Table binding properties**

Table Binding	
Ord	history:/betaStation/DamperPosition

Table binding properties include the following:

- ORD

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### **About field editor bindings**

Field editor bindings are used to bind field editor components to an object.

A field editor is a component that is designed to view and edit an object property. Field editors are designed to be laid out on panes and are built with standard widgets like buttons, text fields, check boxes, and so on.

The properties pane for the field editor binding is shown in [Figure 58. Field editor binding properties, page 91](#).

**Figure 58. Field editor binding properties**



Field editor binding properties include the following:

- ORD

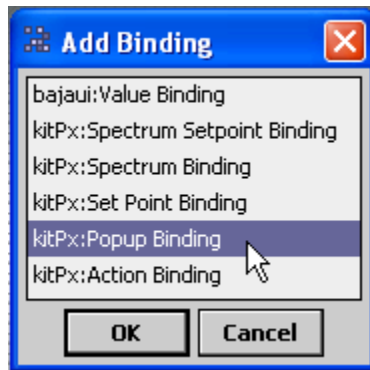
Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### About Popup Bindings

This type of binding may be used to open a Px view in an additional “popup” window that you can specify and configure in terms of size, location, and content.

The Popup binding is attached to a Px object (such as a button) and configured using the Popup Binding properties. The figure below shows the Popup Binding listed in the Add Binding dialog box.

**Figure 59. Popup Action Binding in the Add Binding Dialog box**



The Popup Binding has the following properties:

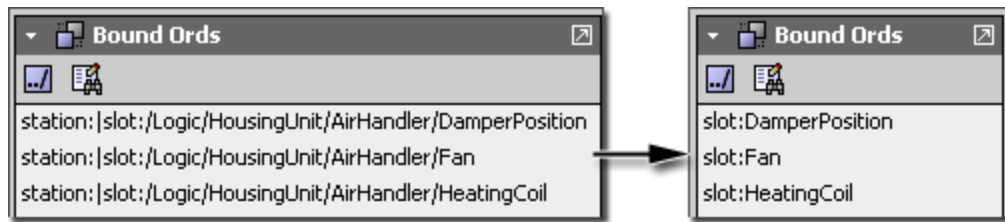
- ord
- Degrade behavior
- title
- position
- size
- modal

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

### About relative and absolute bindings

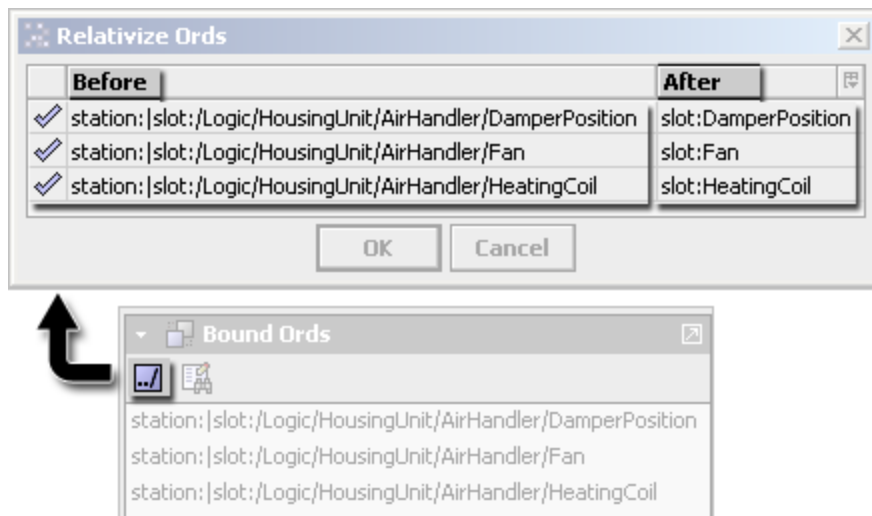
Data bindings, like ORDs, can be relative or absolute.

Since ORDs can be relative or absolute and widgets are bound to data sources using an ORD, data bindings can be relative or absolute, as shown in here.

**Figure 60. Absolutely bound ORDs and relatively bound ORDs**

This is a particularly important point to remember if you want to design Px files that are used in multiple views on a local station or even possibly for use across different stations. When you use the Px Editor tools to bind data, the ORD is usually supplied in an absolute format, by default. The ORD is relative to the station, such as: `station:|slot:/Logic/HousingUnit/AirHandler/DamperPosition`. This absolute path ensures that the data always resolves to a single unique component (“DamperPosition”) in the location that is specified by the ORD, regardless of where the Px file or the parent component is located. If the same Px file is attached to a view that belongs to a different component, the ORD will still resolve to the original “DamperPosition” component because of the absolute path. However, if you make data binding relative, then the path will resolve relative to its current *parent* ORD. This relative path makes the Px file resolve data bindings correctly to identically named components that reside in different locations, thus making one Px file usable in many views.

You can open the **Relativize ORDs** dialog box from the Bound ORDs palette, as shown in [Figure 61. Relativizing bound ORDs, page 92](#). This Px Editor tool provides a convenient way to change absolute ORDs to relative ORDs.

**Figure 61. Relativizing bound ORDs**

Refer to [Types of binding properties, page 84](#) for a description of the binding properties.

## About types of Px target media

When you work in the Px Editor you have a number of client-side target media technologies to consider.

The client-side target media technologies are:

- workbench (WbPxMedia)

The workbench (WbPxMedia) allows the standard Workbench software to run inside a web browser using the Java Plugin. Workbench uses a small applet to download modules as

needed to the client machine and to host the Workbench shell. These modules are cached locally on the browser's hard drive.

- hx (HxPxMedia)

In addition to workbench, a suite of technology called Hx is available. The Hx framework is comprised of a set of server side servlets and a client side JavaScript library. Hx allows a real-time user interface to be built without use of the Java Plugin. It requires only web standards: HTML, CSS, and JavaScript. If you are developing for a target media that is limited to browsers with HTML and CSS with no applet plugins available, be sure to test your Px views often in the appropriate browser as you proceed in your development. The goal of Hx is to satisfactorily present your Px views in non-applet browsers, however, due to the difference in display technologies, you should verify that the Px view displays in the browser as you expect it to.

---

**NOTE:** The capabilities of Hx technology are often enhanced with successive releases of NiagaraAX. An example of this is the “visible” property that can be bound in many widgets.

- The **visible** property functions in Hx exactly as it functions in Px. Previous versions of NiagaraAX did not fully support the **visible** property in Hx.
  - Direct children of Tabbed panes cannot have a hidden visibility (**visible = false**).
  - Direct children of Scroll panes cannot have a hidden visibility (**visible = false**).
  - The **visible** property updates every 5 seconds.
- 

- report (ReportPxMedia)

Report Px pages are simply Px pages that have a Report Pane at the root level, for convenience. Report panes include a Page number Timestamp property in a single column table. See [About Reporting, page 109](#) for more information about reports and the report pane.

- mobile (MobilePxMedia)

Mobile Px pages are designed to enhance and enable widgets for presentation on a mobile device. You must select this option for your Px page if you expect to use the page on a mobile browser. See [About Reporting, page 109](#)

See also, [Choosing a Px Media type, page 51](#).

## About profiles

Profiles provide NiagaraAX software engineers with the ability to customize both the desktop Workbench and the Web Workbench interface.

---

**NOTE:** In Workbench, the concept of profiles pertains to the different ways that Workbench is displayed while using either the Java plugin, Hx, or Mobile technology. In this context, it does not refer directly to security settings or personal preferences.

---

The type of profile used to login to the station can result in certain views being inaccessible even if the user has permissions to access the view.

---

**NOTE:** If you include a workbench view (wb view) as part of a Px page, then from a user-profile perspective it is just a Px page which is allowed. The user permissions to that included workbench view (component) in the station determines whether it displays in the px page or not.

---

Using NiagaraAX, the software engineer can create customized Workbench applications that provide different functionality. Refer to *NiagaraAX Developer Guide* for more information about custom user interface development. Workbench comes with the following two categories of profiles that have specific profile types available:

- Web Profiles (refer to [About Web Profiles, page 94](#))
- Kiosk Profiles (refer to [About Kiosk Profiles, page 101](#))

### **About Web Profiles**

Custom user interface design, at the browser level, allows for different views that may include or exclude features such as sidebars, navigation trees and other tools that are provided through the interface. Web profiles are used to identify these different web interfaces.

You can assign a default Web profile to each user that is listed in the User Manager. In addition, if you have Mobile licensed, you can assign a Default Mobile Web Profile as well. Refer to “User-Service” in the *NiagaraAX User Guide* for details about assigning web profiles to users.

Following are some of the standard Web profile options provided in NiagaraAX:

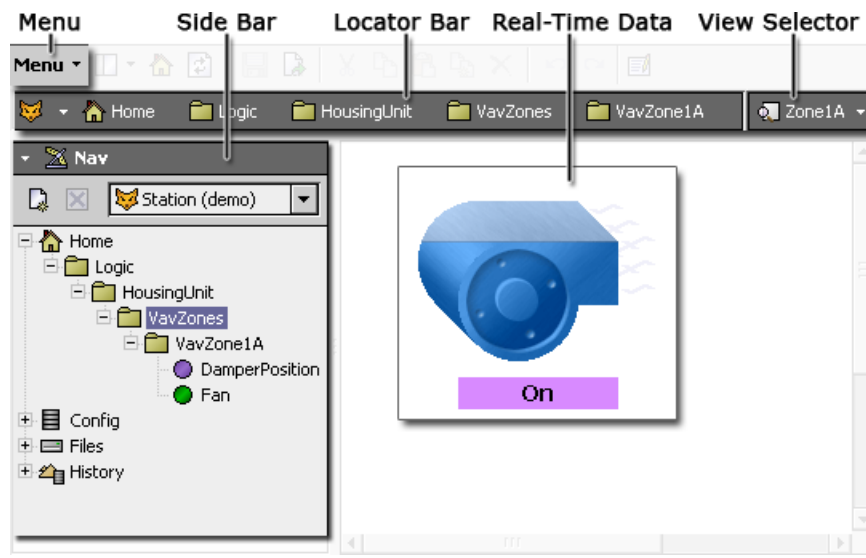
- [Default Wb Web Profile, page 94](#)  
Includes all Workbench functions
- [Simple Admin Wb Web Profile, page 95](#)  
Reduced feature set in Workbench interface but with admin access available
- [Basic Wb Web Profile, page 96](#)  
Reduced feature set in Workbench interface
- [Default Mobile Web Profile, page 97](#)  
The Default Mobile Web Profile provides a way for small, hand-held devices to effectively display views from a NiagaraAX station.
- [Default Hx Profile, page 97](#)  
No Java plugin
- [Basic Hx Profile, page 99](#)  
No Java plugin with reduced feature set interface
- [Default Touchscreen Profile, page 101](#)  
This profile uses Hx technology and provides a real-time user interface without the Java plugin download.
- [Basic Touchscreen Profile, page 100](#)  
This profile uses Hx technology to provide a real-time user interface without the Java plugin download.
- [Handheld Touchscreen Profile, page 101](#)  
This profile uses Hx technology to provide a set of features that are optimized for viewing on a handheld touchscreen device.

### **Default Wb Web Profile**

This profile provides all the features of the Web Workbench, using the full Java plugin download.

An example of the Default Wb Web Profile is show below.



**Figure 62. Default Wb Web Profile example**

Major features of the Default Wb Web Profile include:

- Dropdown menu list

This menu includes the following submenus: **File, Edit, Search, Tools, SideBars, PxEditor**. These menus provide commands that are explained in “Types of menu bar items”, in the *NiagaraAX User Guide*.

- Side Bar

You may show or hide this pane. It may include the **Nav** side bar and the **palette** side bar. The Nav side bar displays the navigation hierarchy that is defined by the nav file.

- Locator Bar

This interactive graphic bar appears across the top of the view area and functions as explained in *NiagaraAX User Guide*.

- Real-Time Data

Data is displayed in real-time using the Java plugin.

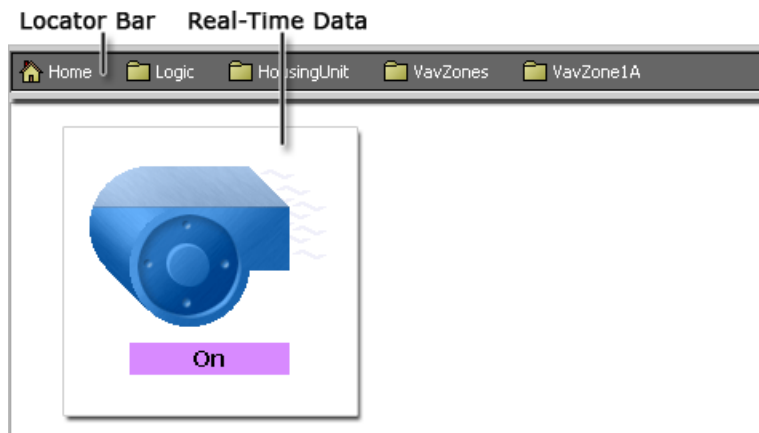
- View selector

Provides the same view selection options that are available in the desktop Workbench views. Refer to *NiagaraAX User Guide* for details.

### Simple Admin Wb Web Profile

This profile provides all the access of the Default Wb Web Profile, *without* using the full Java plugin download. It provides the same user interface as the Basic Wb Web profile but allows the full admin access as the Default Wb Web profile.

An example of Simple Admin Wb Web Profile display is show below.

**Figure 63. Simple Admin Wb Web Profile example**

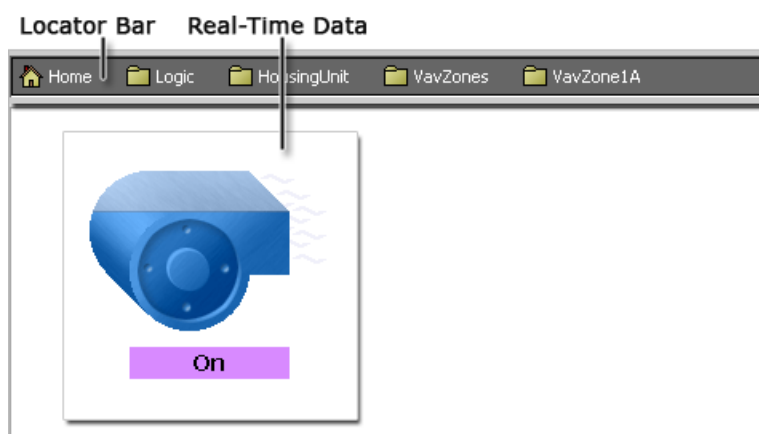
Major features of the Simple Wb Web Profile include:

- Access  
Access to all views (as allowed by user permissions).
- Locator Bar  
This graphic bar appears across the top of the view area – it is not restricted to the nav file.
- Real-Time Data  
Data supplied in real time using the Java plugin.

### Basic Wb Web Profile

This profile uses a reduced set of features but provides a rich user interface using the Java plugin download.

An example of the Basic Wb Web Profile is show below.

**Figure 64. Basic Wb Web Profile example**

Major features of the Basic Wb Web Profile include:

- Locator Bar  
This graphic bar appears across the top of the view area.
- Real-Time Data

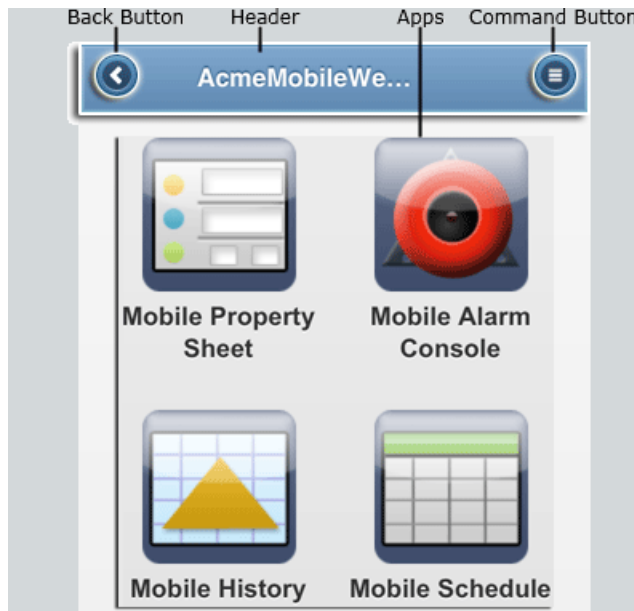
Data supplied in real time using the Java plugin.

### Default Mobile Web Profile

The Default Mobile Web Profile gives you a mobile-friendly option for presentation of NiagaraAX station views on a hand-held device.

Mobile apps use Bajascript technology (see About Bajascript in the *NiagaraAX Mobile Guide*) and require no Java-plugin download. This profile provides a reduced set of features (compared to the Java-plugin profiles) but includes most of the frequently used controls and field editors in a way that is easy to use on a cell phone or tablet. Refer to the *NiagaraAX Mobile Guide* for more details.

**Figure 65. Default Mobile Profile example**



Major features of the Default Mobile Profile include:

- Header navigation
  - The Header Bar appears across the top of the view area and displays a **Back** button and a **Command** button for navigation. This bar can be disabled (hidden) if desired.
- Real-Time Data
  - Data is displayed in real time using Bajascript technology and the NiagaraAX **BOX service** instead of the Java plugin.
- View Selector
  - A limited set of view selections are available through the **Select Views** menu under the **Command** button.

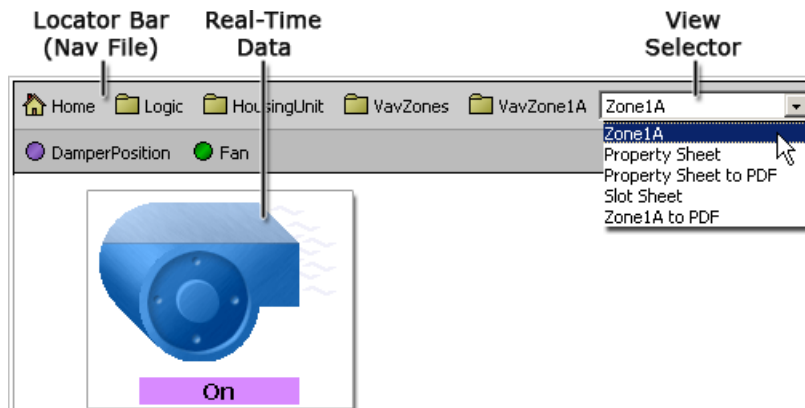
Refer to the *NiagaraAX Mobile Guide* for more details.

### Default Hx Profile

This profile uses Hx technology and provides a real-time user interface without the Java plugin download.




An example of the Default Hx Web Profile is shown below.

**Figure 66. Default Hx Profile example**

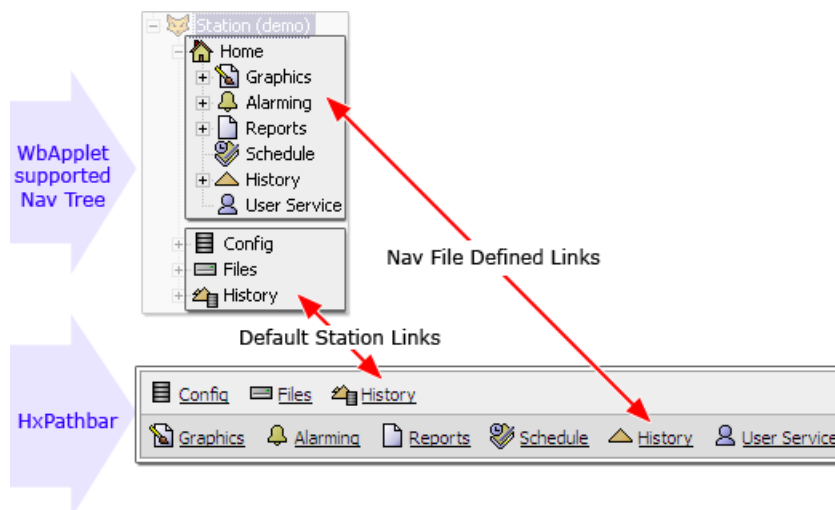


Major features of the Default Hx Profile include:

- **Locator Bar**  
This graphic bar appears across the top of the view area and displays links that are defined by the nav file.
- **HxPathbar**  
The HxPathBar provides a default set of links between the three root spaces: Station (Config)

-  , Files
-  , and History
-  . Users can fully navigate a station in Hx without having to manually type in URLs.

**Figure 67. Comparing the HxPathbar and the WbApplet Nav Tree**



- **Real-Time Data**  
Data is displayed in real time using Hx technology instead of the Java plugin.

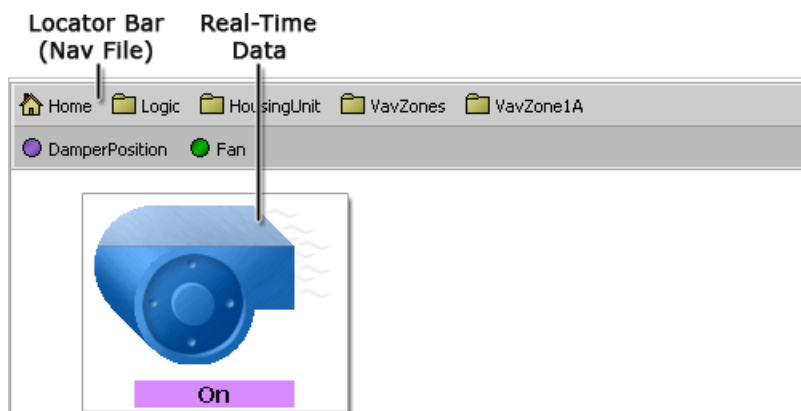
- View Selector This dropdown option list provides a set of alternative views for the active object.
- The **visible** property displays and behaves the same in Hx profiles and WbWeb profiles (using the WbApplet). An example of this is the **visible** property that may be bound and controlled in some widgets and the true/false values set to show or hide a widget based on the **visible** property value. For more details, refer to [Use the visible property in Hx profile \(an example\), page 39](#)

### Basic Hx Profile

This profile uses Hx technology. It provides a reduced set of features but includes real-time user interface without the Java plugin download.

An example of the Basic Hx Profile is shown below.

**Figure 68. Basic Hx Profile example**



Major features of the Basic Hx Profile include:

- Locator Bar

This graphic navigation bar appears across the top of the view area and displays links that are defined by the nav file.

- HxPathbar

The HxPathBar provides a default set of links between the three root spaces: Station (Config)



, Files

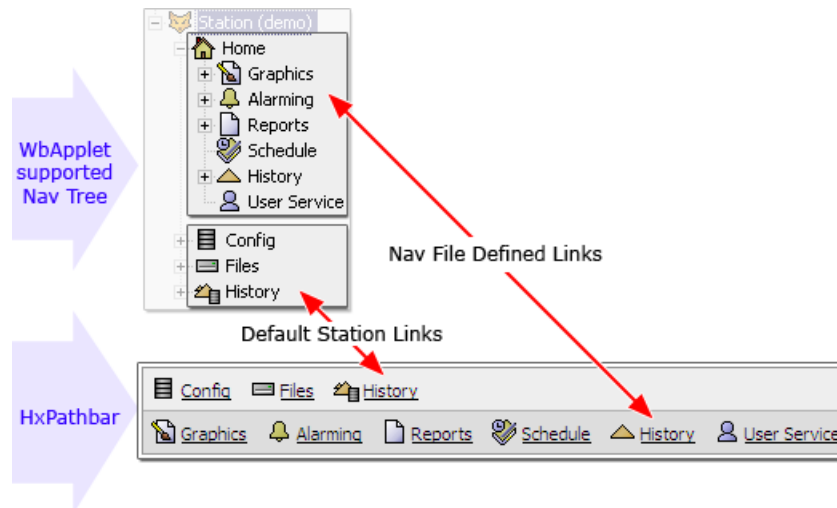


, and History



. Users can fully navigate a station in Hx without having to manually type in URLs.

**Figure 69. Comparing the HxPathbar and the WbApplet Nav Tree**



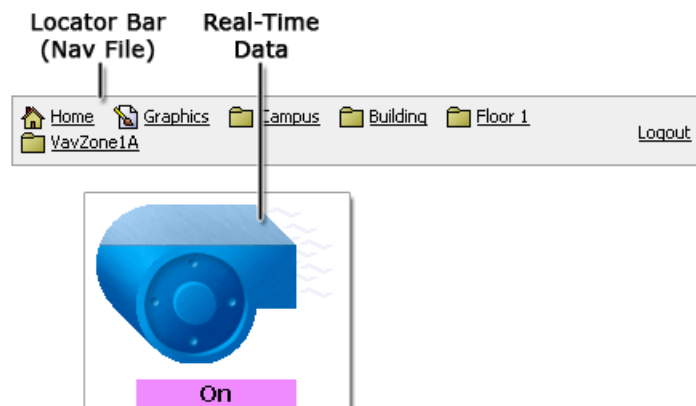
- Real-Time Data  
Data supplied in real time without using Hx technology instead of the Java plugin.
- The **visible** property displays and behaves the same in Hx profiles and WbWeb profiles (using the WbApplet). An example of this is the **visible** property that may be bound and controlled in some widgets and the true/false values set to show or hide a widget based on the **visible** property value. For more details, refer to [Use the visible property in Hx profile \(an example\), page 39](#)

### Basic Touchscreen Profile

This profile uses Hx technology. It provides a reduced set of features but includes real-time user interface without the Java plugin download.

An example of the Basic Touchscreen Profile is shown here:

**Figure 70. Basic Touchscreen Profile example**



Major features of the Basic Touchscreen Profile include:

- Locator Bar  
This graphic bar appears across the top of the view area and displays links that are defined by the nav file.
- Real-Time Data

Data is displayed in real time using Hx technology instead of the Java plugin.

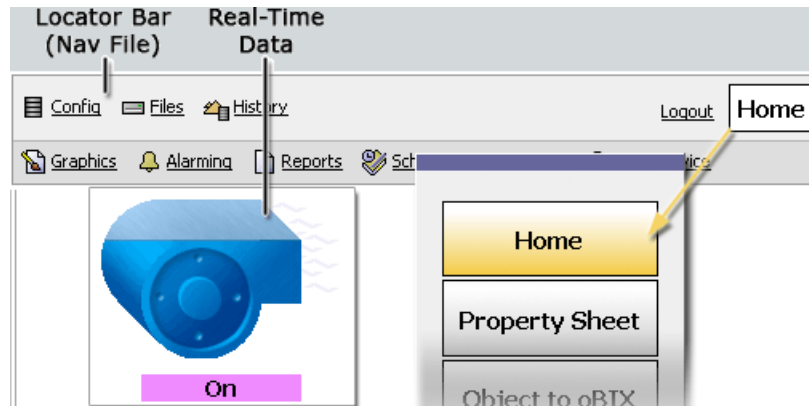
See *NiagaraAX Touchscreen Profile Engineering Notes* document for more details about this profile

### Default Touchscreen Profile

This profile uses Hx technology and provides a real-time user interface without the Java plugin download.

An example of the Default Touchscreen Web Profile is shown below.

**Figure 71. Default Touchscreen Profile example**



Major features of the Default Touchscreen Profile include:

- **Locator Bar**  
This graphic bar appears across the top of the view area and displays links that are defined by the nav file. The Default Touchscreen profile includes the `Config`, `Files`, and `History` space links, in addition to links specified by the assigned nav file.
- **Real-Time Data**  
Data is displayed in real time using Hx technology instead of the Java plugin.
- **View Selector - touchscreen-friendly**  
When touched, this control provides an option dialog box of alternative views for the active object.

See *NiagaraAX Touchscreen Profile Engineering Notes* document for more details about this profile

### Handheld Touchscreen Profile

You can use the Handheld Touchscreen Profile to help set up and optimize views that are going to be displayed on a touchscreen handheld devices.

This profile uses Hx technology (described in [About types of Px target media, page 92](#)). It provides a set of features that are optimized for viewing on a handheld touchscreen device.

See *NiagaraAX Touchscreen Profile Engineering Notes* document for more details about this profile

### About Kiosk Profiles

Kiosk mode provides a way to run a NiagaraAX workbench station so that it fulfills many of the needs of a stand-alone operator workstation as well as many needs of a touchscreen interface.

In order to automatically start Kiosk mode, you must have a *locally connected display*.

---

**NOTE:** The Kiosk profile is not invoked or displayed by remote browser clients. You cannot use a remote computer with a touchscreen and expect to get the same Kiosk interface.

---

While Kiosk mode is not limited to touchscreen applications, it does provide advantages that make it well suited for use in a touchscreen application. The unique interface designs provided by Kiosk profiles are described in the following related topics.

### Basic Kiosk Profile

This profile is designed for normal operators to use a workbench in Kiosk mode.

Basic Kiosk Profile displays a read-only locator bar as a "bread crumb trail" and provides **Back**, **Forward**, and **Logoff** buttons. This profile disables:

- all side bars
- the entire menu
- the entire toolbar
- all non-admin views

### Default Kiosk Profile

This is the default workbench profile used in Kiosk mode.

Default Kiosk Profile supports all the features of a normal desktop workbench profile except for:

- **File > Close command**
- **File > Exit command**
- **Tools > Credentials Manager**

### Handheld Kiosk Profile

This profile provides a reduced display that is designed to help handheld users view and navigate the application.

## About the Px Editor workflow

There are various ways to work with the Px Editor. Some work sequences may depend on the type of control logic you are working with and some may depend on your Px target media. The following sections describe workflow patterns that may be used in some situations but may not be optimal—or even possible with others.

The Px Editor tools provide many ways to perform a single task. For example, you can edit widget properties in the Properties side bar, or you may double-click on a widget to display a Properties pane in a separate dialog box. Both means of editing the properties have the same effect.

## About the Make Widget wizard

The Make Widget wizard is a convenient aid for quickly adding widgets to the Px file. The Make Widget wizard is available when Px Editor is the active view.

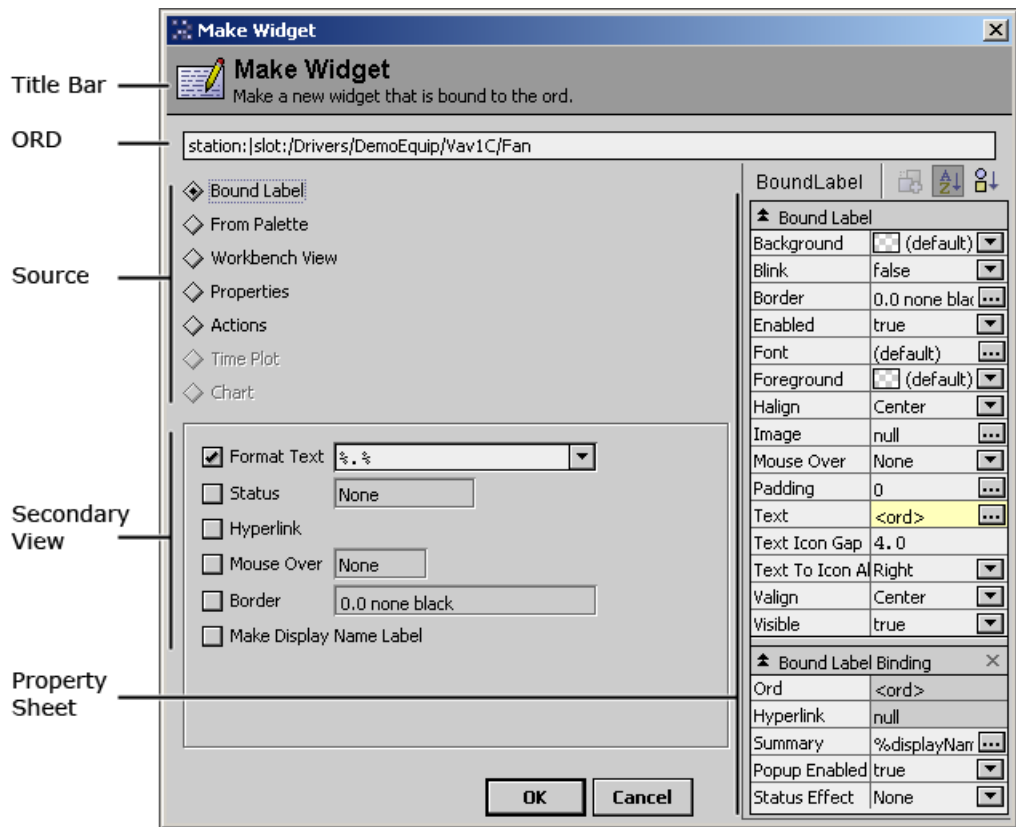
The wizard, shown below, automatically displays when you drag a component onto the Px Editor canvas.

---

**NOTE:** The wizard does not display when you drag a widget object from a palette to the canvas.

---



**Figure 72. Make Widget wizard**

The following list describes the primary areas of the Make Widget wizard:

- **Title Bar**

This area displays a reminder that you are creating a widget that has a value that is bound to the ORD of the object that you just dragged into the Px Editor. You can double-click on the ORD area to browse for a different component.

- **ORD**

This area displays the ORD of the object that you dragged into the Px Editor. You can double click on this line and browse to select a different slot (for example: “out” or “value”) in the current ORD or to bind to a different component.

- **Source**

This area provides context-sensitive options for choosing the type and source of widget that you want to bind to the ORD. Depending on the type of value that you want to link, the wizard displays a variety of options in this area. For example, the Chart widget option would be available for a History Log object but it would appear dimmed (unavailable) if you dragged a Boolean Point object onto the canvas.

The `Widget Source` option that you choose will change the display in the Property Options area and the Property Sheet area.

- **Secondary view**

The secondary view area changes depending on what option is chosen in the Source area:

- **Property Options (Bound Label only)**

This area appears when you select `Bound Label` in the **Widget Source** options. It provides a set of convenience options that allow you to set some of the commonly used label properties, by selecting a check box. You may also use the property sheet area of the wizard or later use the property fields in the property sheet dialog box or property sheet side bar to edit these properties.

- **Module browser**

This area appears when you select the `From Palette` option in the **Source** view and allows you to browse for the desired widget.

- **Sheet selector**

This area appears when you select the `Workbench View` option in the **Source** view and allows you to choose the Workbench view that you want to display for the object you just dragged into the Px Editor.

- **Property options**

This area appears when you select the `Properties` option in the **Source** view and allows you to choose options that relate to the properties of the object that you just dragged into the Px Editor.

---

**NOTE:** The Secondary view does not appear when the `Actions` option is selected in the **Source** area.

---

- **Property Sheet**

This area provides context-dependent options for you to select from (depending on the object that you drop onto the Px Editor).

- **Property Sheet**

The complete property sheet view displays when you use the `Bound Label`, `From Palette`, or `Workbench View` option.

- **Properties List**

A complete list of component properties (for the component you are dropping on the Px Editor canvas) is listed in this area.

- **Actions**

A list of available actions is displayed here when you select the `Actions` option.

## About the Nav file

The Nav file is a special XML file that resides on the file system—not in the station database. It provides navigation in a hierarchical tree structure.

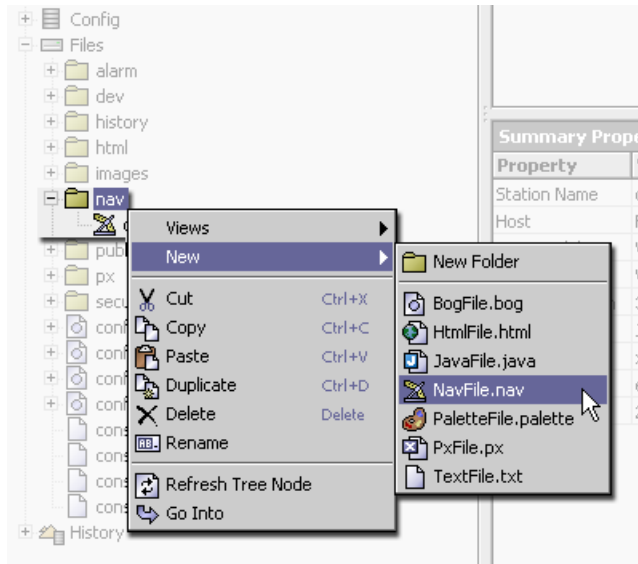
You can create as many Nav files as you want, however, a client may display only one Nav file at any time. This means that any time you are viewing a station you can only see one nav file in the Nav tree (or locator bar) but since a station may be serving many clients, each client may be using different nav files concurrently – based on their nav file assignment. The Nav file is user-specific, so when a user logs into a station, the Nav file for that user is displayed in the Nav tree or locator bar.

You can match a Nav file to a user by using the **Workbench User Manager** view. Refer to the section on “UserService” in the *NiagaraAX User Guide* for details about the User Manager.

You can create new nav files using the right-click menu, as shown below. For more details, see [Creating new nav files, page 43](#).

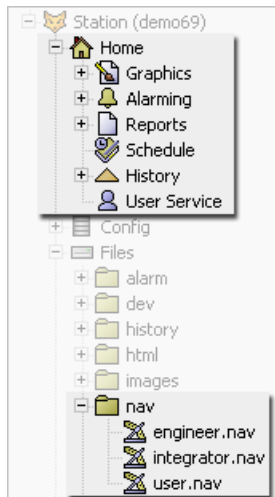
Add details to your nav file and customize the hierarchy of your Nav tree using the **Nav File Editor**, as described in [About the Nav File Editor, page 106](#).

**Figure 73. Creating a new Nav file**



The figure below shows three Nav files in the `nav` folder. Notice that the Nav side bar displays a tree hierarchy of links directly under the station node. Each node of the tree has a context-appropriate icon to help the user recognize the target link of each node in the tree. You can specify icons for the tree nodes, as described in [About the Nav File Editor, page 106](#).

**Figure 74. Nav file sidebar**



### About Nav file elements

You can use nav file elements to create custom nav files.

When you create a new nav file, in addition to the xml declaration, the file includes two default elements, as listed and shown below.

- **<nav> element**  
a single opening and closing set of <nav> elements hold all the nodes in the tree

- **<node> element**

the default <node> element includes an ORD path that points to the station root directory:

```
ord='station:|slot:/'
```

You can edit these nodes directly in the **Text Editor** or you can use the tools provided in the **Nav File Editor** (refer to [About the Nav File Editor, page 106](#)).

**Figure 75. Default nav file with home node only**

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<nav version='1.0'>
  <node name='Home' ord='station:|slot:/' icon='module://icons/x16/home.png'>
  </node>
</nav>
```

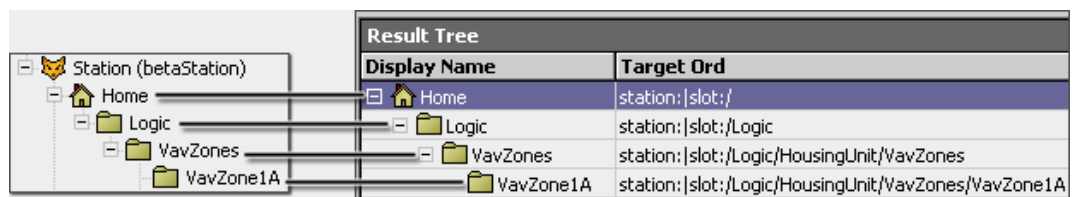
The following graphic shows what the elements in your nav file look like (when viewed in a text editor) once you add a few nodes to your tree.

**Figure 76. Simple nav file with home node only**

```
<?xml version="1.0" encoding="UTF-8"?>
<nav version='1.0'>
  <node name='Home' ord='station:|slot:/' icon='module://icons/x16/home.png'>
    <node name='Logic' ord='station:|slot:/Logic' icon='module://icons/x16/folder.png'>
      <node name='VavZones' ord='station:|slot:/Logic/HousingUnit/VavZones'
        icon='module://icons/x16/folder.png'>
        <node name='VavZone1A' ord='station:|slot:/Logic/HousingUnit/VavZones/VavZone1A'
          icon='module://icons/x16/folder.png' />
      </node>
    </node>
  </node>
</nav>
```

The following graphic shows what the elements in a nav file look like in the Nav tree and in the **Nav File Editor** once you add a few nodes to your tree.

**Figure 77. Nav file viewed in Nav tree and in Nav File Editor (Result Tree)**

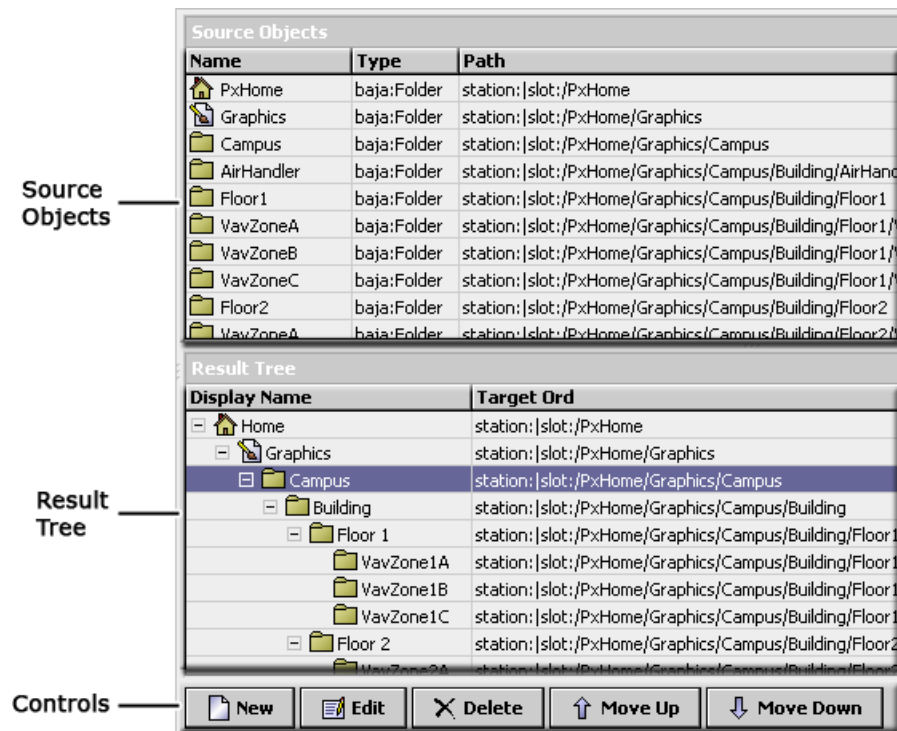


## About the Nav File Editor

The Nav File Editor view provides a simple way for you to view and edit Nav files.

When you select the Nav File Editor view of a file, the Nav File Editor displays in the Workbench view pane.

The default view of the Nav File Editor comprises three main areas, as shown and described below:

**Figure 78. Nav File Editor**

- **Source objects pane**

This resizable pane displays in the top half of the Nav File Editor when you toggle **ON** the **Show Components** button or the Show Files button on the control pane. The pane disappears when you toggle the buttons **OFF**. Refer to [About Nav File Editor source objects, page 107](#) for details about source objects.

- **Result tree pane**

This pane displays whenever the Nav File Editor is open. The pane appears in the bottom half of the Nav File Editor when you toggle **ON** the **Show Components** button or the Show Files button on the control pane and fills the view (above the control buttons) when the **Show Components** button or the Show Files are toggled **OFF**. Refer to [About Nav File Editor result tree, page 108](#) for details about the result tree.

- **Controls**

These buttons are always displayed horizontally across the bottom of the Nav File Editor. Refer to [About Nav File Editor buttons, page 109](#) for more information about the functions of each Nav File Editor control button.

### **About Nav File Editor source objects**

Nav File source objects are those objects that may be used for creating your nav file.

In order to appear in the source object pane, the “source object” must be a Px file or a component with a Px view assigned to it. The source objects populate the pane view based on the position of the **Show Components** and **Show Files** buttons, as described in [About Nav File Editor buttons, page 109](#).

**Figure 79. Source objects available in the source pane**

Source Objects			28 objects
Name	Type	Path	
VavZone1A	baja:Folder	station: slot:/Logic/HousingUnit/VavZones/VavZone1A	
VavZones	baja:Folder	station: slot:/Logic/HousingUnit/VavZones	
Logic	baja:Folder	station: slot:/Logic	
AirHandler.px	file:PxFFile	file:^px/building/AirHandler.px	
Building.px	file:PxFFile	file:^px/building/Building.px	
FloorPlan.px	file:PxFFile	file:^px/building/FloorPlan.px	

To use the source objects pane for creating the nav file structure, you drag components or files from the source objects pane down to desired tree location in the result tree pane. Refer to [About Nav File Editor result tree, page 108](#) for details about the result tree pane.

### About Nav File Editor result tree

The result tree pane is where you build the nav file structure.

You can drag and drop components and files from the source objects pane to create new nodes in your tree. You can also create nodes directly, using the **New** button or by using the popup menu directly in the result tree pane. Refer to [Editing nav files, page 43](#) for procedures about editing nav files.

**Figure 80. Result tree pane**

Result Tree		4 objects
Display Name	Target Ord	
Home	station: slot:/	
Logic	station: slot:/Logic	
VavZones	station: slot:/Logic/HousingUnit/VavZones	
VavZone1A	station: slot:/Logic/HousingUnit/VavZones/VavZone1A	

The figure below shows an example of a source object being dragged to a specific location in the Nav tree. The result tree shows the exact hierarchy of your Nav tree and allows you to set the indent levels by where you drop the component.

**Figure 81. Dragging a source object to the result tree pane**

Source Objects		
Name	Type	Path
VavZone1A	baja:Folder	station: slot:/Logic/HousingUnit/VavZones/VavZone1A
VavZones	baja:Folder	station: slot:/Logic/HousingUnit/VavZones
Logic	baja:Folder	station: slot:/Logic
AirHandler.px	file:PxFFile	file:^px/building/AirHandler.px

Result Tree	
Display Name	Target Ord
Home	station: slot:/
Logic	station: slot:/Logic
VavZones	station: slot:/Logic/HousingUnit/VavZones
VavZone1A	

## About Nav File Editor buttons

The Nav File Editor buttons are aligned horizontally across the bottom of the Nav File Editor. They include controls that you use to create, edit, delete, and rearrange nodes in the nav file.

The following list describes each of the of the Nav File editor buttons.

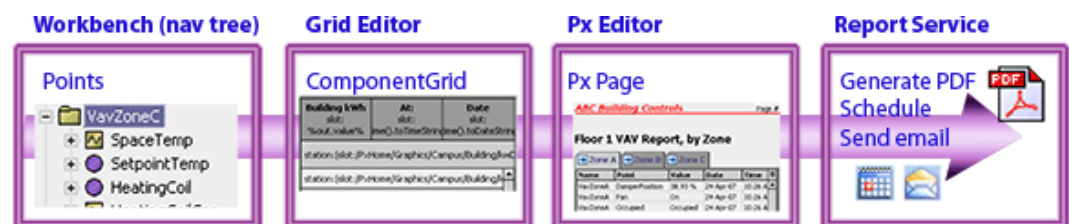
- **New**  
This button displays the **New Node** dialog box that allows you to create a new node in the nav file tree by specifying a name, target ORD and icon for the node. The new node is created as a child node to the active point in the tree (the one you have selected).
- **Edit**  
This button is available when you select a node in the Result Tree pane. It displays the **Edit Node** dialog box that allows you to edit the node name, target ORD, and icon.
- **Delete**  
This button deletes a selected node in the Result Tree pane.
- **Move Up**  
This button allows you to move a selected node (or nodes) up in the nav file tree hierarchy.
- **Move Down**  
This button allows you to move a selected node (or nodes) down in the nav file tree hierarchy.
- **Show Components**  
This button toggles on and off. Toggle this button **ON** to display all eligible components in the Source Objects pane.
- **Show Files**  
This button toggles on and off. Toggle this button **ON** to display all eligible Px files in the Source Objects pane.

## About Reporting

The Reporting function in NiagaraAX helps you design, display, and deliver data to online views and to printed pages. Table layout features and report widgets are used in designing the reports. The report service and the report export feature allow you to schedule and distribute reports by email at any time. NiagaraAX Workbench views and services help you create and route reports.

The figure below shows the primary views that facilitate the reporting process.

**Figure 82. Primary views used in creating and routing reports**



The following topics describe the major concepts and views associated with reporting.

- [Reporting process workflow, page 45](#)  
This section describes the major points in the reporting process.
- [Types of report components, page 110](#)

This section describes components that are located in the Reports module and available in the Reports palette.

- [About the Grid Table view, page 116](#)

This section provides a description of the Grid Table view of the ComponentGrid.

- [About the Grid Label Pane view, page 117](#)

This section provides a description of the Grid Label Pane view of the ComponentGrid.

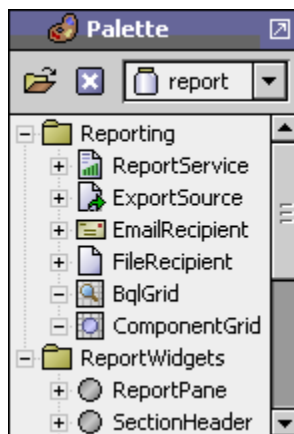
- [About the Grid Editor view, page 119](#)

This section provides a description of the Grid Editor view of the ComponentGrid.

### **Types of report components**

Report components are found in the Report palette.

**Figure 83. Report palette**



Types of report components include the following:

- ReportService  
Refer to [About the ReportService component, page 111](#) for details.
- ExportSource  
Refer to [About the ExportSource component, page 111](#) for details.
- EmailRecipient  
Refer to [About the EmailRecipient component, page 112](#) for details.
- FileRecipient  
Refer to [About the FileRecipient component, page 113](#) for details.
- BqlGrid  
Refer to [About the BqlGrid component, page 113](#) for details.
- ComponentGrid  
Refer to [About the ComponentGrid component, page 114](#) for details.
- ReportPane  
Refer to [About the ReportPane component, page 115](#) for details.
- SectionHeader  
Refer to [About the SectionHeader component, page 116](#) for details.

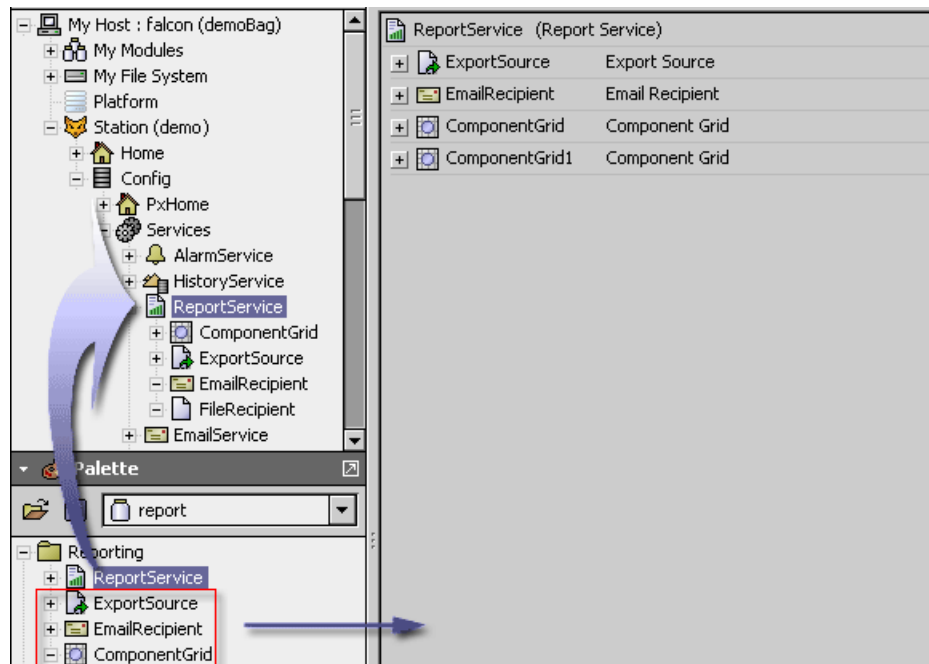


**NOTE:** For more information about using BFormat scripting, refer to the Engineering Note document, *BFormat (Baja Format) Property Usage*.

### About the ReportService component

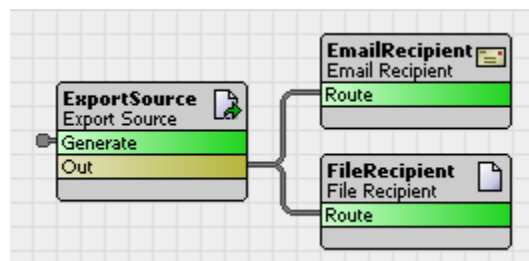
ReportService is the parent, or container, for the ExportSource, EmailRecipient, ComponentGrid, and FileRecipient components. In order to use the report service, copy the ReportService component from the Report module to the Services folder in the Workbench Nav tree, as shown.

**Figure 84.** Copy the report service to the services directory and add report components



You can use the ReportService wire sheet view to link generated reports to the desired recipients, as shown in here.

**Figure 85.** ReportService component Wire Sheet view



### About the ExportSource component

The Export Source component should be located in the ReportService container. It allows you to define a report display source (typically a Px file) for exporting and it allows you to schedule a time that you want to export it.

The property sheet view, shown below, contains the properties that you configure for these functions.

**Figure 86. Export Source component property sheet view**

- **Schedule**

This property contains standard scheduling options that allow you to choose different options for timing the delivery of your report. It also displays a **Last Trigger** and **Next Scheduled Trigger** field.

- **Source**

This property specifies the report display that you want to email. An arrow button located at the right end of the property display opens the **Export Source Wizard** dialog box. This dialog box provides two steps that assist you in assigning source location and selecting a PDF export or an oBix driver export for your report.

### About the EmailRecipient component

The report-EmailRecipient component is found in the Report palette (do not confuse this component with the email-EmailRecipient found in the Email palette) and contains properties that allow you to specify where the report is to be emailed.

The property sheet view, shown below, displays the properties that you configure for this function.

**Figure 87. Email Recipient component property sheet view**

EmailRecipient component properties include the following:

- **To**

This is the email address of the email recipient (person who is to receive the emailed report).

- **Cc**

This is the email address of anyone that is to receive a copy of the email (and attached report).

- **Bcc**

This is the email address of anyone that should receive a copy of the email (and attached report) but whose name you do not want to display in the delivered email.

- **Language**

This is the ISO 639 language code as two lower-case letters. For a list of codes, see the following: <http://www.loc.gov/standards/iso639-2/langcodes.html>

- **Email Account**

This is an option list that allows you to choose the email account that you want to use for sending the report email.

### About the FileRecipient component

The FileRecipient component allows you to save a report to a location under a station file system.

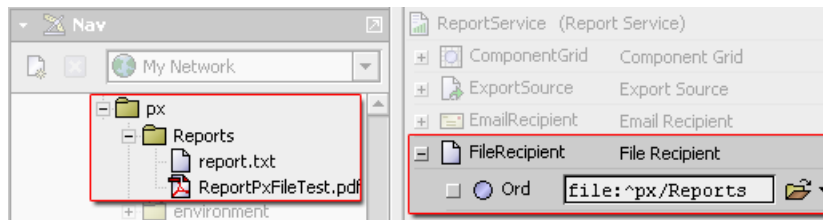
This component contains a single property that allows you to specify a location (file directory) for saving the report. The property sheet view, shown below, displays the property that you configure for this function.

---

**CAUTION:** Saving excessively to a flash drive can reduce drive life prematurely. Do not “over-schedule” report saves to a flash drive. Saving to a hard drive does not have such limitations.

---

**Figure 88. FileRecipient component specifying location for generated report file**



FileRecipient component properties include the following:

- **Ord**

This property provides a text field that allows you to specify the directory for saving the generated report file.

### About the BqlGrid component

The BqlGrid component provides a means for using a Bql Query and the BFormat syntax to define a dynamic table layout.

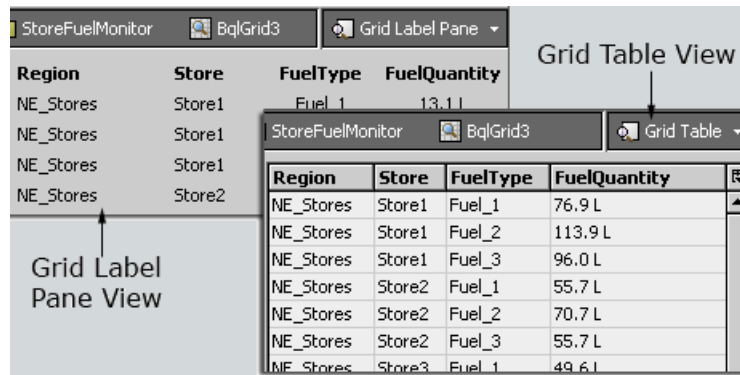
Use the BqlGrid component to specify an OrdTarget and columns of values to build a dynamic Grid Table or Grid Label Pane view. The BqlGrid component subscribes to the specified values for dynamic updating.

---

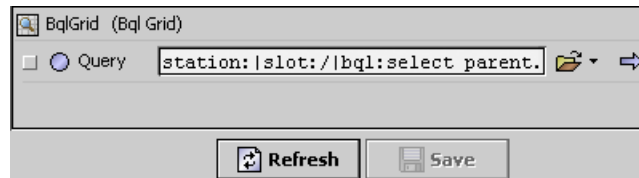
**NOTE:** This does not work with histories.

---

In addition to the standard wire sheet, category sheet, slot sheet, and property sheet views, the BqlGridComponent property includes a Grid Table and Grid Label Pane view, as shown here:

**Figure 89. BqlGrid Label Pane view**

The BqlGrid component property sheet view is shown here and described below:

**Figure 90. BqlGrid property sheet view**

- **Query**

This single property is used to define a TargetOrd using a Bql query. Every column in the resultant row points to the same Ord. A value for each desired column in the row is provided by using BFormat syntax to point to the corresponding column from the Bql query. This is different from the ComponentGrid component (see [About the ComponentGrid component, page 114](#)) where each table cell is its own OrdTarget.

---

**NOTE:** For more information about using BFormat scripting, refer to the Engineering Note document, *BFormat (Baja Format) Property Usage*.

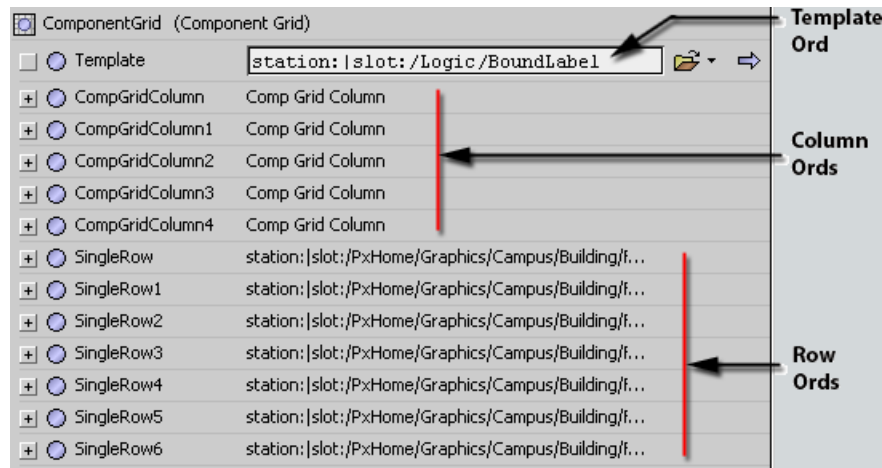
---

### About the ComponentGrid component

The component grid allows you to define and design your report content (typically in the Grid Editor view).

Use the component grid to link and layout the data that you want to put into your report.

**Figure 91. ComponentGrid property sheet view**



The component grid has most of the standard Workbench views, such as Property Sheet view (shown above), wire sheet, category sheet, slot sheet, and link sheet. Unique ComponentGrid views include the following:

- **Grid Table**

This view displays your linked data in a typical Workbench table format. Refer to [About the Grid Table view, page 116](#) for details.

- **Grid Label pane**

This view displays your linked data in an tabular view that uses the GridLabelPane widget for displaying the report data. Refer to [About the Grid Label Pane view, page 117](#) for details.

- **Grid Editor**

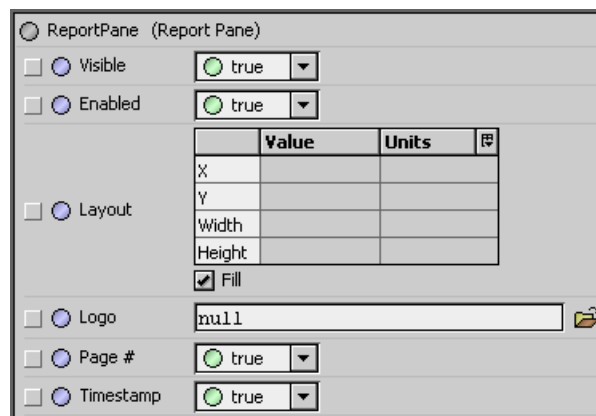
This is the view that you use to specify row and column data for your report. Refer to [About the Grid Editor view, page 119](#) for details.

### About the ReportPane component

The ReportPane widget is located in the ReportWidgets folder and provides a container for layout of Px page report views.

The ReportPane has the standard Visible, Enabled, and Layout properties, but also has the following unique properties:

**Figure 92. ReportPane property sheet view**



- **Logo**

This property supports the linking of a graphic to the ReportPane component. When displayed in a Px page, or a Report, the graphic originates at the top left corner of the page. Type or browse to the desired graphic to set the file path in the property field. If no logo is used, the default value of “null” should be set in the property field.

- **Page#**

When this property value is set to `True`, page numbering displays on each Report page.

- **Timestamp**

When this property value is set to `True`, a timestamp is displayed in the bottom right corner of the report pages.

## About the SectionHeader component

The SectionHeader component is used to put a title or heading at the top of a report.

**Figure 93. Section header property sheet**

## About the Grid Table view

The Grid Table view is a view of the ComponentGrid component. This view provides a tabular display of all the points that are assigned to the ComponentGrid.

In the Grid Table view, you can write to points (issue commands) using the popup menu, if you have write permissions. Points in the table also display color-coded status, as shown.

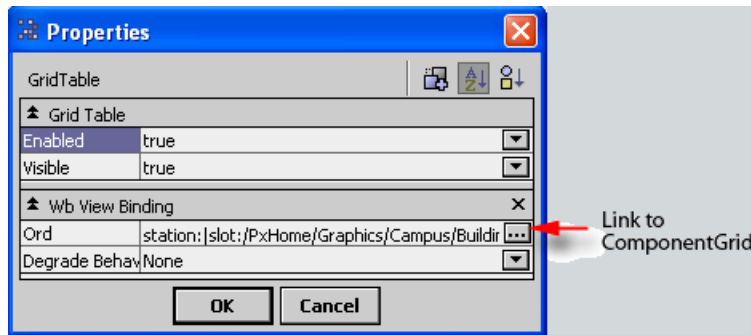
**Figure 94. Grid Table view**

Name	Point	Value	Date	Time	
VavZoneA	DamperPosition	70.79 %	23-Apr-07	8:04 AM	
VavZoneA	Fan	On	23-Apr-07	8:04 AM	
VavZoneA	Occupied	Occupied	23-Apr-07	8:04 AM	
VavZoneA	AirFlow	603.9	23-Apr-07	8:04 AM	
VavZoneA	HeatingMode	Off	23-Apr-07	8:04 AM	
VavZoneA	HeatingCoil	70.79 %	23-Apr-07	8:04 AM	
VavZoneA	SetpointTemp	72.00 °F	23-Apr-07	8:04 AM	

- Emergency Override
- Emergency Auto
- Override
- Auto
- Set

The table controls and options are similar those that are used in most Workbench table controls (see *NiagaraAX User Guide*).

When you use a GridTable view in a Px page, the editable table properties are accessed in the Properties dialog box, as shown below:

**Figure 95. Grid Table properties in Px Editor**

- **Enabled**

When set to `true`, the table in the Px page interface is “commandable” using the popup menu. When this property is set to `false`, the display is still visible but not commandable.

- **Visible**

When set to `true`, the table in the Px page interface is visible. When this property is set to `false`, the display is not visible.

- **Ord**

This Ord is the link binding the display to the GridComponent that defines the table.

- **Degrade Behavior**

These options specify how the table behaves when the binding communications are not available.

### About the Grid Label Pane view

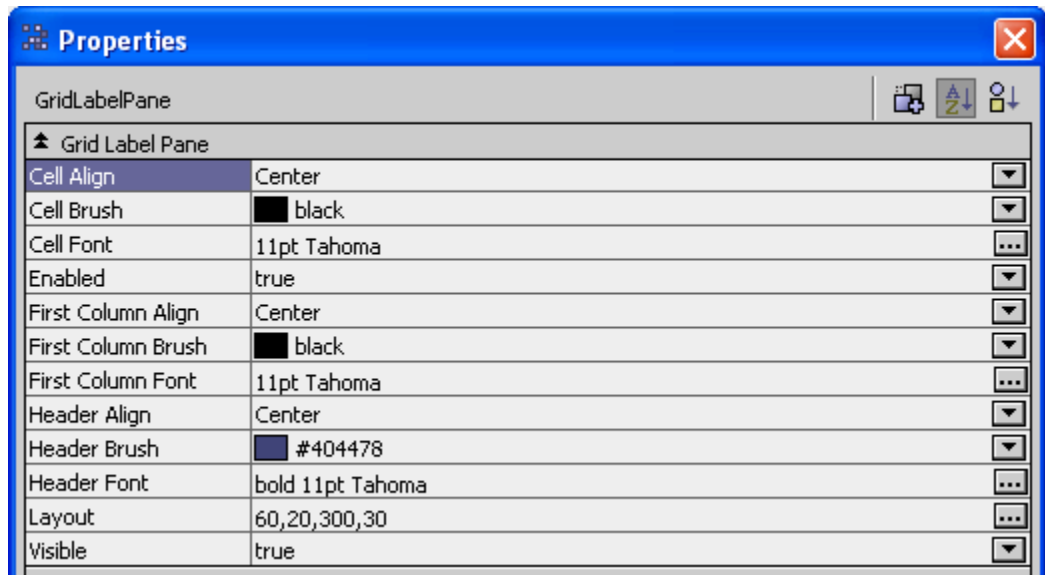
The Grid Label Pane view is a view of the ComponentGrid component. This view provides a tabular display of all the points that are assigned to the ComponentGrid.

In the Grid Table Pane view, you can write to points (issue commands) using the popup menu, if you have write permissions. Points in the table also display color-coded status, as shown.

**Figure 96. Grid table view of the GridComponent**

Name	Point	Value	Date	Time
VavZoneA	DamperPosition	82.78 %	30-Apr-07	5:06 PM
VavZoneA	Fan	On	30-Apr-07	5:06 PM
VavZoneA	Occupied	Occupied	30-Apr-07	5:06 PM
VavZoneA	AirFlow	663.9	30-Apr-07	5:06 PM
VavZoneA	HeatingMode	On	30-Apr-07	5:10 PM
VavZoneA	HeatingCoil	82.78 %	30-Apr-07	5:06 PM
VavZoneA	SetpointTemp	88.00 °F	30-Apr-07	5:06 PM
VavZoneA	SetpointTemp	88.00 °F	30-Apr-07	5:06 PM
VavZoneA	SetpointTemp	88.00 °F	30-Apr-07	5:06 PM
VavZoneA	SetpointTemp	88.00 °F	30-Apr-07	5:06 PM
VavZoneA	HeatingCoilGen	82.8	30-Apr-07	5:06 PM
VavZoneA	Occupied	Occupied	30-Apr-07	5:06 PM
VavZoneA	Fan	On	30-Apr-07	5:06 PM

In the Px Editor view, you can use the GridLabelPane widget to layout the table on a Px page and edit the properties of the display properties, as shown in the figure below. These properties allow you to change the font-type and color, as well as set the text alignment of the table cells, table heading, and first column properties.

**Figure 97. Grid Label properties in Px editor**

GridLabelPane properties include the following:

- **Cell Align**  
This property provides options for setting the alignment of cell content (left, right, center, fill).
- **Cell Brush**  
This property specifies the color of the content (for example, font color) within the table cells.
- **Enabled**  
When set to `true`, the table in the Px page interface is “commandable” using the popup menu. When this property is set to `false`, the display is still visible but not commandable.
- **First Column Align**  
This property provides options for setting the alignment of cell content (left, right, center, fill) for the first column in the table.
- **First Column Brush**  
This property specifies the color of the content within the first column table cells.
- **Header Align**  
This property provides options for setting the alignment of cell content (left, right, center, fill) for the top row of the table (the table “heading”).
- **Header Brush**  
This property specifies the color of the content within the table heading cells
- **Header Font**  
This property allows you to set the font-type and font properties for the table heading.
- **Layout**  
This property allows you to specify the size and location of the table in relation to its parent (X, Y, coordinates, height and width). Alternatively, you can select a “fill” option to have the table expand to the limits of its parent container.



- **Visible**

When set to `true`, the table in the Px page interface is visible. When this property is set to `false`, the display is not visible.

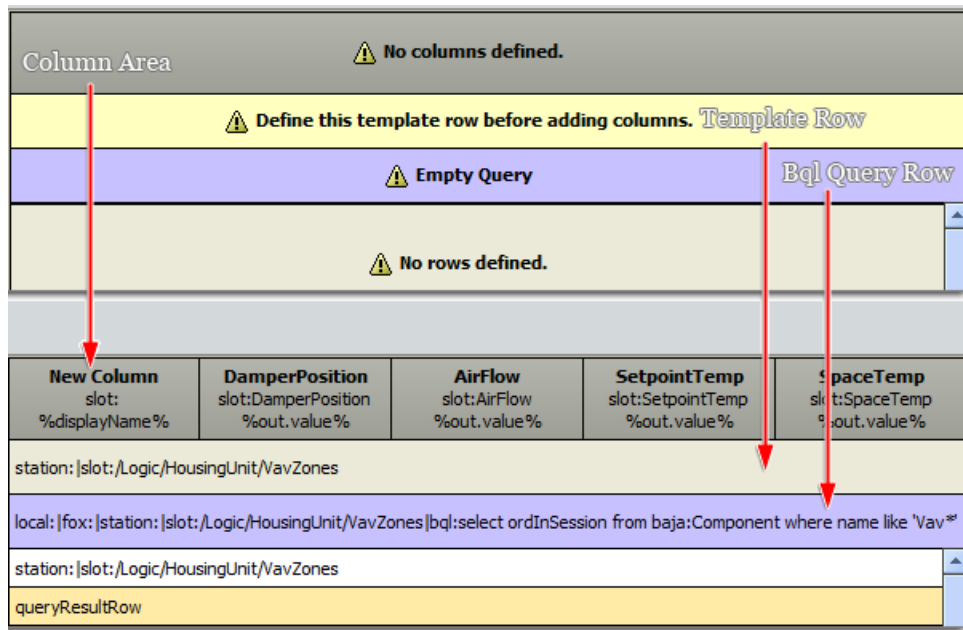
### About the Grid Editor view

The Grid Editor view is one view of the ComponentGrid component. This view provides a layout display that you use to assign points that you are going to use in a report. The figure below shows the Grid Editor view with and without points assigned.

[About the Report Edit Column/Edit Row dialog boxes, page 121](#)

In the Grid Editor view, you can assign components to areas using the **Add Column** and **Add Row** commands from the popup menu or from the Workbench main menu. You can edit existing rows or columns by double-clicking on them to display the **Edit Row** or **Edit Column** dialog box, as shown here:

**Figure 98. Adding rows and columns in the Grid Editor view**



- **Columns area**

This area extends across the top of the Grid Editor view. Columns specify the slot path that identifies the point data that goes in the table cell. Add columns *after* you have defined a template row in the template area.

- **Template area**

This area is just below the column area and displays as yellow, with a caution message when it is empty. The template row is used as a reference for picking columns, and therefore, there must be a template row defined before you can add and save a column in the Grid Editor view. Double-click on the template area to add a row using the **Edit Template** dialog box.

**NOTE:** If you add a row without first adding a Template Row, then the first row you add becomes the Template Row by default and automatically populates the template area.

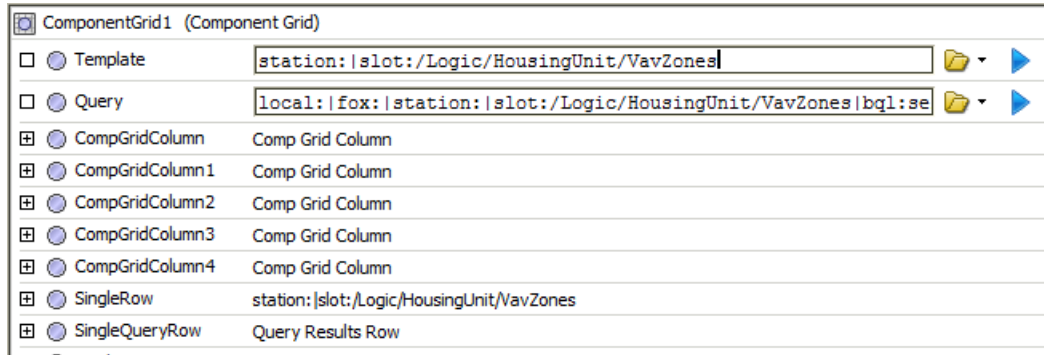
- **Row area**

The row area extends across the lower part of the Grid Editor view. Add rows to define any point data that goes in the table row. Rows work in conjunction with the columns to present data in a table format.

- **Bql Query**

This field allows you to enter a Bql query to resolve rows in the grid. This feature improves the performance of the BqlGrid component for resolving rows. It ensures reliable point subscription for those points defined by the BqlQuery.

**Figure 99. Edit Query available starting in NiagaraAX-3.7**



For more details on using the Grid Editor view, see [About using the Grid Editor view, page 120](#)

### About using the Grid Editor view

The typical workflow for using the Grid Editor view is:

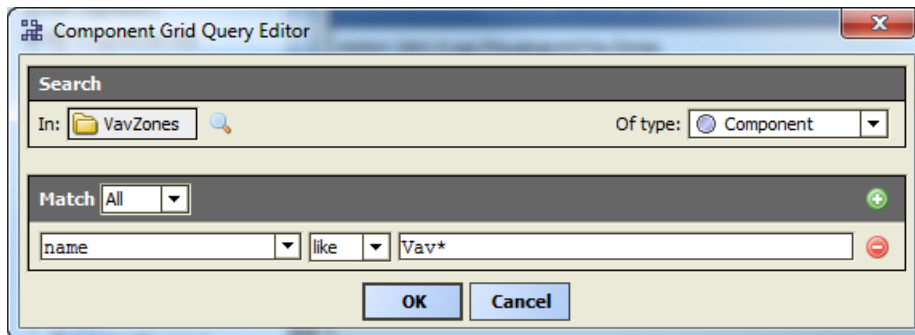
- Add a template.  
Drag a component from the Nav tree area onto the yellow “Template Row”.
- Add a row  
You can double-click directly on the “Empty Query” row to use the **Component Grid Query Editor**.

---

**NOTE:** Using a query (as opposed to dragging and dropping a component on the row) provides better point subscription service to points.

---

**Figure 100. Component Grid Query Editor**



- Add columns

Use the **Add Column** button on the main toolbar to add desired columns in the Grid Editor.

### About editing rows/columns in a ComponentGrid

In the **Grid Editor** view, you can assign components to areas using the **Add Column** and **Add Row** commands from the popup menu or from the Workbench main menu. You can edit existing rows or columns by double-clicking on them to display the **Edit Row** or **Edit Column** dialog box, as shown.

---

**NOTE:** Add columns *after* you have defined a template row in the template area.

---

Step 1. In the Grid Editor view, assign a component to a column or row:

- right-click in the Column Area and select **Add Column** from the popup menu.
- right-click in the Row Area and select **Add Row** from the popup menu.

Depending on your selection in step 1, either the **Edit Column** or **Edit Row** dialog box displays.

Step 2. If adding a column you can enter the following details in the **Edit Column** dialog box:

- **Name**

Enter a name to assign and display name/title to the column

- **Format**

Enter literal text or BFormat scripting.

- **Ord**

Displays in **Edit Column** and **Edit Row** dialog boxes. Designate a point by browsing or entering the ord of the component that holds the value that you want to display.

---

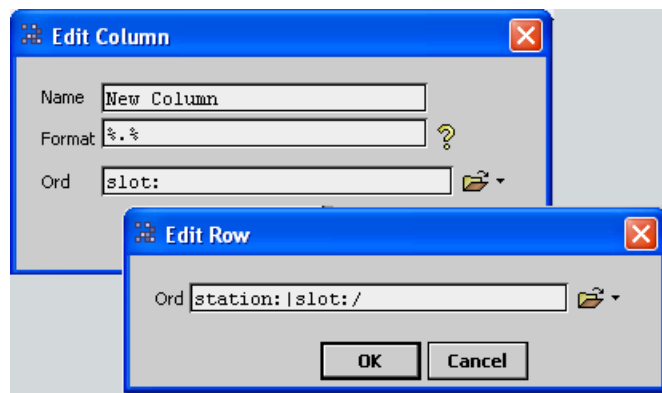
**NOTE:** If assigning rows, you can drag and drop a point onto the Grid Editor view and the Ord automatically appears in the Ord field.

---

### About the Report Edit Column/Edit Row dialog boxes

The **Edit Row** or **Edit Column** dialog boxes display when you add a new row or column—or when you choose to edit an existing row or column in the Grid Editor view.

**Figure 101. Edit Row and Edit Column dialog boxes**



The following fields are associated with these edit dialog boxes:

- **Name**

This field is associated with the **Edit Column** dialog box. Type in a literal name to assign a display name or title to the column.

- **Format**

This field is associated with the **Edit Column** dialog box. You can use literal text and BFormat scripting to assign a value to the column.

- **Ord**

This field displays in both the **Edit Column** dialog box and the **Edit Row**.

Designate a point by browsing to or typing the Ord of the component that holds the value that you want to display.

---

**NOTE:** For assigning rows, you can drag and drop a point onto the Grid Editor view and the Ord automatically appears in the Ord field.

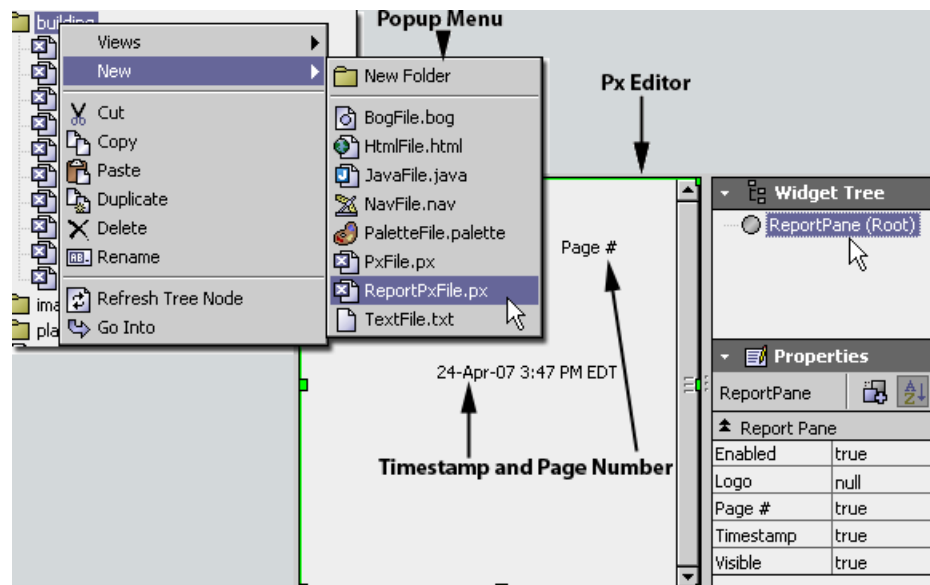
---

### About the ReportPxFile

ReportPx files are available from the **New** menu item on a popup menu when you right-click on an appropriate container for a Px file.

The ReportPxFile is a regular Px file that has been “pre-loaded” with a ReportPane widget, as shown here.

**Figure 102. Creating a new ReportPxFile**



### About Mobile

Mobile applications (apps) and the supporting Niagara environment are designed to work with modern browsers on a personal computer, a mobile phone, or other hand-held devices.

Mobile applications (apps) are supported and included in the Niagara Framework. These apps and the supporting Niagara environment are designed to work with modern browsers on a personal computer, a mobile phone, or other hand-held devices.

**Figure 103. Niagara Mobile Web Apps example device browser view**



Mobile apps have the potential to run on a variety of platforms, including: iPhone, Android or Blackberry operating systems.

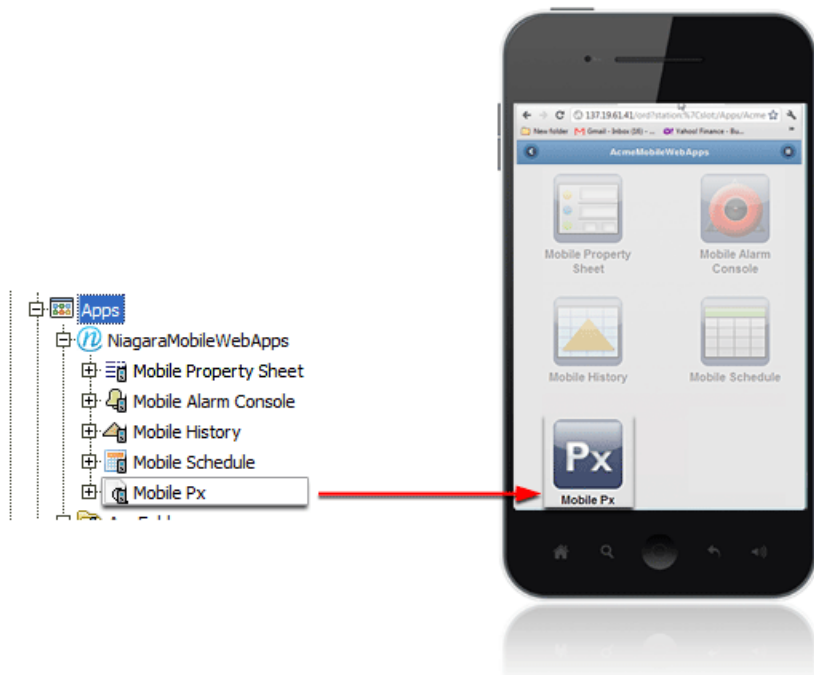
The primary value of mobile web apps is their ability to present users with information in a hand-held browser. However, before the user can view, edit, and navigate between these apps, the apps must be licensed, installed, and configured on a station using NiagaraAX Workbench. The apps that are provided with Workbench are easy to install and configure using a simple set of characteristic NiagaraAX properties. For more information refer to the *NiagaraAX Mobile Guide*.

### **About Mobile Px App**

You can use the Mobile Px app to create your own app views in the Px Editor. You can use the standard Niagara Px Editor to design views with navigation and custom controls that display and work well in a hand held device.

The Mobile Px app allows you to create your own app views in the Px Editor, without having to use any software programming language. This includes the ability to use the standard Niagara Px Editor to design views with navigation and custom controls that display and work well in a hand held device. Essentially, you can think of this app as being an “app builder” itself.

**Figure 104. Mobile Px app in Workbench and Mobile Desktop view**



Mobile Px provides a way for you to present live data and field editors in expandable lists that are designed to work well with mobile devices. It is not for absolutely positioning graphics or other media on the page, such as you would for designing a visual layout of a controls system. So, when you create views with this Mobile Px, the display is not an exact representation of what you have in the Px Editor or what you would expect to see when you are displaying views created in Hx. Instead, for Mobile Px views, the Px Editor works as a tool to create views that the Mobile Px app interprets and presents using such “mobile-friendly” features as: rounded buttons, “liquid” layout, dynamic views, and others.

However, many of the standard Px graphic widget and property options are still available for configuring your Px page, including:

- **Buttons**

Add standard buttons (Action Button, Hyperlink Button, Save Button) from the kitPx palette. These buttons work the same but appear rounded in mobile views.

- **Labels and bound labels**

Add standard labels from the kitPx palette.

- **Field editors**

- **Graphic alignment**

- **Font family**

- **Font size and style**

For more information about Mobile Px, refer to the *NiagaraAX Mobile Guide*.

# CHAPTER 4 COMPONENT GUIDES

## TOPICS COVERED IN THIS CHAPTER

- **Components in bajai module**
- **Components in chart module**
- **Components in gx module**
- **Components in kitPx module**
- **Components in pxeditor module**
- **Components in report module**

The Component Guides provide summary information on common graphics components.

## Components in bajai module

The following are components in the **bajai** module:

### ***BorderPane***

#### **bajai-*BorderPane***

*BorderPane*, in the **bajai** palette, provides border content for a widget similar to the CSS Box Model. *Margin* defines the space between the bounds and the border. *Padding* defines the space between the border and the child bounds. *Border* defines the look and width of the border around each of the sides.

### ***Bookmark***

*Bookmark* is a convenience feature supported by the *bajai-Bookmark* component.

#### **bajai-*Bookmark***

*Bookmark*, found in the **bajai** palette, provides a quick reference link to a specified “bookmark” location.

### ***BoundTable***

#### **bajai-*BoundTable***

*BoundTable*, in the **bajai** palette, is a *Table* implementation that is ready to use with *Table-Bindings* to populate its model.

### ***Button***

#### **bajai-*Button***

*Button*, in the **bajai** palette, is a control button which fires an action when clicked.

### ***CanvasPane***

#### **bajai-*CanvasPane***

*CanvasPane*, in the **bajai** palette, is commonly used to provide absolute layout using *Layout* for most *Widgets* and *Geom* for *Shapes*. *CanvasPane* also provides support for scaling and/or aligning its contents.

The CanvasPane component also is available in the **mobile** palette. It provides support for absolute positioning for Widgets in a Mobile Px page. When used in a Mobile Px file, the CanvasPane **scale** options are limited to `None` or `Fit Ratio`. For more details, see the section on “About the Mobile Px app” in the *NiagaraAX Mobile Guide*.

## **CheckBox**

### **bajai-CheckBox**

CheckBox, in the **bajai** palette, is a specialized ToggleButton which displays its label next to a box which can be checked and unchecked.

## **ConstrainedPane**

### **bajai-ConstrainedPane**

ConstrainedPane, in the **bajai** palette, is a pane with a constrained size. It allows the pane to be restricted by minimum width, minimum height, and maximum height.

## **EdgePane**

### **bajai-EdgePane**

EdgePane, in the **bajai** palette, is a container with a layout like the `java.awt.BorderLayout`. It only supports five potential children in the frozen slots top, bottom, left, right, and center. The top and bottom widgets fill the pan horizontally use their preferred height. The left and right widgets use their preferred width and occupy the vertical space between the top and bottom. The center widget gets all the remaining space. Any of the widgets may be a `NullWidget`.

## **Ellipse**

### **bajai-Ellipse**

Ellipse, in the **bajai** palette, renders a ellipse geometry.

## **ExpandablePane**

### **bajai-ExpandablePane**

ExpandablePane, in the **bajai** palette, contains two widgets. A summary widget is displayed all the time, with a button to the right used to expand and collapse the pane. When expanded, the expansion widget is displayed under the summary.

## **FlowPane**

### **bajai-FlowPane**

FlowPane, in the **bajai** palette, lays out its children from left to right and top to bottom fitting as many children as possible in each row. It is based on the `java.awt.FlowLayout`.

## **GridPane**

### **bajai-GridPane**

GridPane, in the **bajai** palette, provides flexible layout based on a grid with a predefined number of columns. Cells are laid out left to right, flowing to the next row based on the `columnCount` property. Row height is determined by the max preferred height of the widgets in the row. If `uniformRowHeight` is true then all row heights are sized using the max row. Column width is determined by the max preferred width of the widgets in the column. If



`uniformColumnWidth` is true then all column widths are sized using the max column. If `rowLayout` is set to fill then all widgets in a given row are sized to fill the row height. Otherwise the widgets use their preferred height and are aligned accordingly. If `columnLayout` is set to fill then all widgets in a given column are sized to fill the column width. Otherwise the widgets use their preferred width and are aligned accordingly. The `columnGap` field specifies how many pixels to leave between columns. The `rowGap` field specifies how many pixels to leave between rows. By default if the actual space of the pane is bigger than the preferred size, then the `horizontalPaneAlignment` and `verticalPaneAlignment` fields determine where to put the extra space. Or the `stretchColumn` and `stretchRow` properties may be used to indicate that a specific column or row should be stretched to fill the additional space. Enabling the stretch feature trumps pane alignment. If the `colorRows` field is true then alternating rows are shaded to produced a striped effect.

## ***HyperlinkLabel***

### **bajai-HyperlinkLabel**

The `HyperlinkLabel`, in the **bajai** palette, has the same properties as the `Label` (see “`bajai-Label`”), plus an `ORD` property that allows you to assign an `ORD` to the label. When an `ORD` path is supplied for this property, the `HyperlinkLabel` causes a mouse cursor to change to a standard link cursor and the component performs a hyperlink when clicked.

## ***Label***

### **bajai-Label**

`Label`, in the **bajai** palette, is used to display text and/or an icon. Labels are always transparent, so their background is defined by their parent. Their preferred size is always an exact fit to contain the text and/or icon. Labels may be used by themselves to display information which cannot respond to user input, or they may be embedded in widgets such as `Buttons`.

## ***Line***

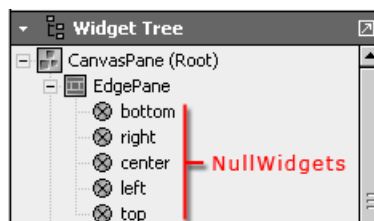
### **bajai-Line**

`Line`, in the **bajai** palette, renders a line between two points.

## ***NullWidget***

### **bajai-NullWidget**

The `NullWidget`, in the **bajai** palette, is a place holder widget to use as a null property value. For example, `NullWidgets` are used in an `EdgePane` widget to hold a place for the Top, Left, Center, Right, and Bottom contents, as shown below:



## ***Path***

### **bajai-Path**

Defines a general path to draw or fill, using a model and string format based on the SVG path element. This component is in the **bajai** palette.

## ***Picture***

### **bajai-Picture**

The Picture widget, in the **bajai** palette in AX-3.8, provides scaling functionality. A `scale-Mode` property allows you to stretch and skew an image to fit within the Picture widget's borders. A Picture can be backed with any image, including JPGs and PNGs, but you will get the most benefit out of Picture scaling when you use SVGs images. For more details, refer to [About SVG support, page 62](#).

## ***Polygon***

### **bajai-Polygon**

Models a closed area defined by a series of line segments, using string format “x,y,width,height”. This component is in the **bajai** palette.

## ***PxInclude***

### **bajai-PxInclude**

This widget provides a way for users to embed a single Px file in another Px file. The PxInclude widget is located in the **bajai** palette. For more details, refer to “About the PxInclude Widget”. For more details, see [About the PxInclude Widget, page 76](#).

## ***RadioButton***

### **bajai-RadioButton**

RadioButton, in the **bajai** palette, is a specialized ToggleButton which displays its label next to a circle which can be checked and unchecked. It is used with groups of other RadioButtons to provide a choice that is exclusive of other options.

## ***Rect***

### **bajai-Rect**

Rect, in the **bajai** palette, renders a rectangle shape.

## ***ScrollPane***

### **bajai-ScrollPane**

ScrollPane, in the **bajai** palette, layouts one child using a viewport which may be scrolled to view the entire child's actual size.

## ***Separator***

### **bajai-Separator**

Separator, in the **bajai** palette, is used in ToolBars and Menus to provide a visible separator between groups of buttons.

## ***Slider***

### **bajai-Slider**

Slider, in the **bajai** palette, provides a visual slider which is used to select an integer or floating point value between a fixed range.

## ***SplitPane***

### **bajai-SplitPane**

SplitPane, in the **bajai** palette, is a pane with a divider between two child widgets. The children can be laid out horizontally or vertically. Use the SplitPane properties to configure the behavior and position of the divider between the two panes.

## ***TabbedPane***

### **bajai-TabbedPane**

TabbedPane, in the **bajai** palette, is a container which displays a set of LabelPane children one at a time using a set of “tabs” to select the currently displayed child. The LabelPane’s label is used to label each tab, and the content of the current selection fills the rest of the pane.

## ***TextEditorOptions***

### **bajai-TextEditorOptions**

The TextEditorOptions, in the **bajai** palette, stores the options used to configure text entry. These are stored under `/user/{user}/textEditor.options`. The Status Colors options allow you to view and change the following (default):

- Show Spaces (false)
- Show Tabs (false)
- Show Newlines (false)
- Tab To Space Conversion (2)
- Show Margin (0)
- Color Coding
  - Foreground (00 00 00 black)
  - Whitespace (c0 c0 c0 gray)
  - Number Literal (80 00 80 purple)
  - String Literal (80 00 80 purple)
  - Identifier (00 00 00 black)
  - Keyword (00 00 ff blue)
  - Preprocessor (80 00 00 maroon)
  - Bracket (ff 00 00 red)

- Line Comment (00 80 00 green)
- Multi Line Comment (00 80 00 green)
- Non Javadoc Comment (80 80 80 dark gray)
- Key Bindings (with default)
  - Move Up Up
  - Move Down Down
  - Move Left Left
  - Move Right Right
  - Page Up PageUp
  - Page Down PageDown
  - Line Start Home
  - Line End End
  - Document Start Ctrl + Home
  - Document End Ctrl + End
  - Word Left Ctrl + Left
  - Word Right Ctrl + Right
  - Cut Ctrl + X
  - Copy Ctrl + C
  - Paste Ctrl + V
  - Undo Ctrl + Z
  - Redo Ctrl + Alt + Z
  - Delete Delete
  - Backspace Backspace
  - Cut Line Ctrl + Y
  - Delete Word Ctrl + Delete
  - Tab Forward Tab
  - Tab Back Shift + Tab
  - Toggle Slash Slash Esc
  - Word Wrap Ctrl + W
  - Goto Line Ctrl + G
  - Find F5
  - Find Next Ctrl + F
  - Find Prev Ctrl + Shift + F
  - Replace F6
  - Reload Macros Ctrl + M
  - Select All Ctrl + A
- Undo Navigation (true)
- Match Parens (true)

- Match Braces (true)
- Match Brackets (true)

### ***ToggleButton***

#### **bajoui-ToggleButton**

ToggleButton, in the **bajoui** palette, provides a two-state widget which may be selected and unselected.

### ***ValueBinding***

#### **bajoui-ValueBinding**

ValueBinding, in the **bajoui** palette, is used to bind to Values typically under a Component. Widget properties may be overridden by creating dynamic slots of the same name with a Converter that maps the bound target to the property value.

## **Components in chart module**

### ***BarChart***

#### **chart-BarChart**

One of two chart types available (the other is Linechart).

### ***ChartCanvas***

#### **chart-ChartCanvas**

chartCanvas is the canvas widget under a ChartPane.

### ***ChartHeader***

#### **chart-ChartHeader**

ChartHeader is the header widget under a ChartPane.

### ***ChartPane***

#### **chart-ChartPane**

ChartPane is the container widget created when adding a Px chart.

### ***DefaultChartLegend***

#### **chart-DefaultChartLegend**

DefaultChartLegend is the legend widget under a ChartPane.

### ***LineChart***

#### **chart-LineChart**

One of two chart types available (the other is BarChart).

## Components in gx module

### ***Brush***

#### **gx-Brush**

Brush is used to change presentation. the Brush is available in the **gx** module.

## Components in kitPx module

### ***ActionBinding***

#### **kitPx-ActionBinding**

kitPx-ActionBinding invokes an action on the binding target component when an event is fired by the parent widget. The ORD of an action binding must resolve down to a specific action within a component.

Additional information and related topics include:

- [About action bindings, page 90](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

### ***AnalogMeter***

#### **kitPx-AnalogMeter**

kitPx-AnalogMeter is a widget that is designed to look and act like a analog meter. This component comes with a value binding that should be bound to a value by an ORD.

Additional information and related topics include:

- [About Widgets, page 65](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

### ***Bargraph***

#### **kitPx-Bargraph**

kitPx-Bargraph is a widget that is designed to display a single value in a bar graph. In addition to action bindings, value and setpoint bindings are available.

Additional information and related topics include:

- [About value bindings, page 87](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

To plot more than one value using a single widget (using a line chart), you can use a chart widget, as described in [Make Chart widgets, page 20](#) . Or, you can use a chart widget for history tables, as described in the *NiagaraAX User Guide* section "About the history chart view".

## ***BackButton***

### **kitPx-BackButton**

kitPx-BackButton is designed for “Touchscreen” displays that require a larger button. This widget works like the standard workbench “Back” button that is located in the top left corner of the workbench user interface. The button is dimmed when no previous page has been visited and is enabled when a previous page is available to go back to when you click it. It is located in the kitPx palette and is pre-configured so that all you have to do is drag it onto a Px page and save the page for it to take effect.

Additional information and related topics include:

- [ButtonGroup, page 133](#)
- [ExportButton, page 134](#)
- [ForwardButton, page 134](#)
- [LogoffButton, page 136](#)

## ***BoundLabel***

### **kitPx-BoundLabel**

kitPx-BoundLabel is a widget that simply provides a label widget (see “bajau-Label” on page 4-83) with a binding already available.

Additional information and related topics include:

- [About bound label bindings, page 86](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## ***BoundLabelBinding***

### **kitPx-BoundLabelBinding**

kitPx-BoundLabelBinding is used exclusively for connecting a value to a bound label widget. Bound labels, which are located in the kitPx module, have properties that you can edit and access from the Px Editor properties side bar.

Additional information and related topics include:

- [About bound label bindings, page 86](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## ***ButtonGroup***

### **kitPx-ButtonGroup**

kitPx-ButtonGroup is a widget that provides a convenient way to bind a group of buttons to a common boolean or enum point. This widget is comprised of a standard GridPane with a “Button-GroupBinding” already available. The ButtonGroup properties are editable in the Px Editor properties side bar.

Additional information and related topics include:

- [ButtonGroupBinding, page 134](#)
- [ExportButton, page 134](#)

- [ForwardButton](#), page 134
- [LogoffButton](#), page 136
- [Types of data bindings](#), page 83
- [Types of binding properties](#), page 84

### ***ButtonGroupBinding***

#### **kitPx-ButtonGroupBinding**

kitPx-ButtonGroupBinding is used exclusively for connecting a value to a ButtonGroup widget. ButtonGroups, which are located in the kitPx module, have properties that you can edit and access from the Px Editor properties side bar.

Additional information and related topics include:

- [ButtonGroup](#), page 133
- [ExportButton](#), page 134
- [ForwardButton](#), page 134
- [LogoffButton](#), page 136
- [Types of data bindings](#), page 83
- [Types of binding properties](#), page 84

### ***ExportButton***

#### **kitPx-ExportButton**

kitPx-ExportButton can be used in “Touchscreen” displays that require a larger button. It works the same way as the Export button that is located on the workbench toolbar menu (opens the Export dialog box when clicked). The ExportButton widget is located in the kitPx palette and is pre-configured so that all you have to do is drag it onto a Px page and save the page for it for it to take effect.

Additional information and related topics include:

- [BackButton](#), page 133
- [ButtonGroup](#), page 133
- [ForwardButton](#), page 134
- [LogoffButton](#), page 136

### ***ForwardButton***

#### **kitPx-ForwardButton**

kitPx-ForwardButton is designed for “Touchscreen” displays that require a larger button. This widget works like the standard workbench “Forward” button that is located in the top left corner of the workbench user interface. The button is dimmed when no “forward” page is available and is enabled when a page is available to go forward to when you click it. It is located in the kitPx palette and is pre-configured so that all you have to do is drag it onto a Px page and save the page for it for it to take effect.

Additional information and related topics include:

- [BackButton](#), page 133
- [ButtonGroupBinding](#), page 134



- [ExportButton](#), page 134
- [LogoffButton](#), page 136
- [Types of data bindings](#), page 83
- [Types of binding properties](#), page 84

### ***GenericFieldEditor***

#### **kitPx-GenericFieldEditor**

kitPx-GenericFieldEditor is a field editor that allows you to edit properties from a graphic view on non-specific fields. It is designed for layout on panes and is comprised of standard widgets like buttons, text fields, and check boxes. Specialized field editor widgets include: SetPointFieldEditor (see [SetPointFieldEditor](#), page 137).

Additional information and related topics include:

- [About field editor bindings](#), page 90
- [Types of data bindings](#), page 83
- [Types of binding properties](#), page 84

### ***ImageButton***

#### **kitPx-ActionButton (ImageButton)**

kitPx-ActionButton is a button that uses images to display the **state** of the button. For example, the action button may have three separate images to indicate the `normal`, `mouseOver`, and `pressed` states. In addition to other properties, the image button includes the following distinctive properties:

- **Mouse Over**

This property allows you to assign an image that appears when the mouse hovers over the button.

- **Normal**

This property allows you to assign an image that appears when the button is not in a pressed or mouseOver state.

- **Pressed**

This property allows you to assign an image that appears when the button is pressed.

See the following topics for related information about buttons:

- [RefreshButton](#), page 136
- [SaveButton](#), page 136
- [Button](#), page 125
- "bajoui-ToggleButton"

### ***IncrementSetPointBinding***

#### **kitPx-IncrementSetPointBinding**

kitPx-IncrementSetPointBinding is used to increment (increase) or decrement (decrease) a setpoint value.

Additional information and related topics include:

- [About setpoint bindings](#), page 88

- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## **LogoffButton**

### **kitPx-LogoffButton**

kitPx-LogoffButton is designed for “Touchscreen” displays that require a larger button. When clicked, this widget automatically initiates a “logoff” command and “disconnects” the current user from any active session. The Logoff button is located in the kitPx palette and is pre-configured so that all you have to do is drag it onto a Px page and save the page for it to take effect.

Additional information and related topics include:

- [BackButton, page 133](#)
- [ButtonGroup, page 133](#)
- [ExportButton, page 134](#)
- [ForwardButton, page 134](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## **RefreshButton**

### **kitPx-RefreshButton**

kitPx-RefreshButton is used to issue a **refresh** command when it is clicked. The “Refresh” text that appears on the button is included by default in the Text property. This button is designed to work well with “Touchscreen” displays that require a larger button. The Refresh button is located in the kitPx palette and is pre-configured so that all you have to do is drag it onto a Px page and save the page for it to take effect.

Additional information and related topics include:

- [BackButton, page 133](#)
- [ButtonGroup, page 133](#)
- [ExportButton, page 134](#)
- [ForwardButton, page 134](#)

## **SaveButton**

### **kitPx-SaveButton**

kitPx is used to issue a save command when it is clicked. The “Save” text that appears on the button is included by default in the Save property—but, like the other properties, it may be edited, as desired.

See the following topics for related information about buttons:

- [ImageButton, page 135](#)
- [RefreshButton, page 136](#)
- [Button, page 125](#)
- [ToggleButton, page 131](#)

## ***SetPointBinding***

### **kitPx-SetPointBinding**

kitPx-SetPointBinding is used to display the current value of a setpoint and also to provide the ability to modify it. The setpoint binding ORD must resolve down to the specific property that is being manipulated.

Additional information and related topics include:

- [About setpoint bindings, page 88](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## ***SetPointFieldEditor***

### **kitPx-SetPointFieldEditor**

kitPx-SetPoinFieldEditor is a special field editor that is designed to allow you to edit setpoint values from a graphic view. It is designed for layout on panes and is comprised of standard widgets like buttons, text fields, and check boxes. For editing non-specific properties, use the kitPx GenericField-Editor (see [GenericFieldEditor, page 135](#))

Additional information and related topics include:

- [About field editor bindings, page 90](#)
- [Types of data bindings, page 83](#)
- [Types of binding properties, page 84](#)

## **Components in pxeditor module**

### ***NewPxViewDialog***

#### **pxeditor-NewPxViewDialog**

NewPxViewDialog allows you to create a new Px view.

## **Components in report module**

### ***BqlGrid***

#### **report-BqlGrid**

The BqlGrid component allows you to create a Grid Table or Grid Label Pane view of any collection of data that you can access with a Bql query and the BFormat syntax. The BqlGrid component is available in the Reporting folder of the **report** palette. Refer to [About the BqlGrid component, page 113](#) for more details.

### ***ComponentGrid***

#### **report-ComponentGrid**

The ComponentGrid allows you to define and design your report content (typically in the Grid Editor view). Use the component grid to link and layout the data that you want to put into your report. The ComponentGrid component is available in the Reporting folder of the **report** palette. Refer to [About the ComponentGrid component, page 114](#) for more details.

## ***EmailRecipient***

### **report-EmailRecipient**

The EmailRecipient component contains properties that allow you to specify where the report is to be emailed. The property sheet view displays the properties that you configure for this function. The EmailRecipient component is available in the Reports folder in the **report** palette. Refer to [About the EmailRecipient component, page 112](#) for more details.

## ***ExportSource***

The ExportSource component allows you to define a report display source (typically a Px file) for exporting and it allows you to schedule a time that you want to export it.

### **report-ExportSource**

The property sheet view contains the properties that you configure for these functions. This component is available in the **report** palette. Refer to [About the ExportSource component, page 111](#) for more details.

## ***FileRecipient***

The file recipient component allows you to save a report to a location under a station file system.

### **report-FileRecipient**

This component contains a single property that allows you to specify a location (file directory) for saving the report. This component is available in the **report** palette. Refer to [About the FileRecipient component, page 113](#) for more details.

## ***ReportPane***

### **report-ReportPane**

The ReportPane widget is available in the ReportWidgets folder of the **report** palette. It provides a container for layout of Px page report views. The ReportPane has the standard Visible, Enabled, and Layout properties, but also has unique properties. Refer to [About the ReportPane component, page 115](#) for more details.

## ***ReportService***

### **report-ReportService**

The report service is the parent, or container, for the ExportSource, EmailRecipient, and ComponentGrid components. In order to use the report service, copy the ReportService component from the Report module to the Services folder in the Workbench nav tree. It is available in the Reporting folder of the **report** palette. Refer to [About the ReportService component, page 111](#) for more information.

## ***SectionHeader***

### **report-SectionHeader**

The SectionHeader component is used to put a title or heading at the top of a report. It is located in the ReportWidgets folder in the **report** palette. More information about the SectionHeader component is available in [About the SectionHeader component, page 116](#).

# CHAPTER 5 PLUGIN GUIDES

## TOPICS COVERED IN THIS CHAPTER

- **Plugins in pxEditor module**
- **Plugins in report module**

The Plugin Guides provide summary information on views of common graphics components.

There are many ways to view plugins (views). One way is directly in the tree. In addition, you can right-click on an item and select one of its views. Plugins provide views of common graphics components.

## Plugins in pxEditor module

Summary information on views of the PxEditor component.

### ***pxEditor***

The PxEditor provides tools for creating and editing Px files.

### **pxEditor-PxEditor**

The Px editor is the view that is used to create graphic presentations in NiagaraAX.

For more details, refer to the following topics:

- [About Px Editor, page 55](#)
- [Chapter 1 About Presentation XML \(Px\), page 1](#)
- *About the Px Editor menu*

## Plugins in report module

Summary information on views of ComponentGrid component.

### ***GridTable***

The Grid Table view is a view of the **ComponentGrid** component.

### **report-GridTable**

This view provides a tabular display of all the points that are assigned to the ComponentGrid.

For more details, see [About the Grid Table view, page 116](#).

### ***GridLabelPane***

The Grid Label Pane view is a view of the **ComponentGrid** component.

### **report-GridLabelPane**

This view provides a tabular display of all the points that are assigned to the ComponentGrid. In the Grid Table Plane view, you can write to points (issue commands) using the popup menu, if you have write permissions.

For more details, see [About the Grid Label Pane view, page 117](#).

### ***ComponentGridEditor***

The Grid Editor view is a view of the **ComponentGrid** component.

### **report-ComponentGridEditor**

This view provides a layout display that you use to assign points that you are going to use in a report.

For more details, see [About the Grid Editor view, page 119](#).

# INDEX

## A

Action bindings.....	90
Alignment .....	57
Animate	
using Popup Binding.....	6
using static SVGs.....	10
Animating graphics	
about.....	81
AX-3.8	
new features .....	2

## B

bajau	
components.....	125
Basic Kiosk .....	102
Binding properties	
types.....	84
Bindings	
action .....	90
bound label .....	86
field editor.....	90
increment setpoint.....	89
Popup.....	91
relative vs. absolute.....	91
setpoint.....	88
spectrum.....	88
spectrum setpoint.....	89
table .....	90
types.....	83
using Popup Binding.....	6
value .....	87
converters.....	87
Bound label binding	
about.....	86
BqlGrid	
component.....	113

## C

Canvas	
grid	
show hatch	
snap.....	56
Change log .....	iii
chart	
components.....	131
client-side target media.....	92
ComponentGrid	
component.....	114
Components	
bajau .....	125–126
GridPane .....	126
HyperlinkLabel.....	127

Label.....	127
Line .....	127
NullWidget .....	127
Path .....	128
Picture .....	128
Polygon .....	128
PxInclude .....	128
RadioButton.....	128
Rect .....	128
ScrollPane.....	128
Separator.....	129
Slider .....	129
SplitPane .....	129
TabbedPane .....	129
TextEditorOptions .....	129
ToggleButton .....	131
ValueBinding.....	131
Bookmark .....	125
BorderPane .....	125
BoundTable.....	125
Button .....	125
CanvasPane.....	125
chart.....	131
BarChart.....	131
ChartCanvas.....	131
ChartHeader .....	131
ChartPane.....	131
DefaultChartLegend .....	131
LineChart .....	131
Checkbox.....	126
ConstrainedPane.....	126
EdgePane .....	126
Ellipse.....	126
ExpandablePane.....	126
FlowPane .....	126
gx .....	132
Brush .....	132
kitPx.....	132
ActionBinding.....	132
AnalogMeter.....	132
BackButton .....	133
Bargraph.....	132
BoundLabel.....	133
BoundLabelBinding.....	133
ButtonGroup.....	133
ButtonGroupBinding.....	134
ExportButton .....	134
ForwardButton .....	134
GenericFieldEditor .....	135
ImageButton .....	135
IncrementSetPointBinding .....	135
LogoffButton.....	136
RefreshButton .....	136
SaveButton.....	136
SetPointBinding.....	137
SetPointFieldEditor .....	137

- pxeditor ..... 137  
 NewPxViewDialog ..... 137  
 report ..... 111, 137  
 BqlGrid ..... 113, 137  
 ComponentGrid ..... 114, 137  
 EmailRecipient ..... 112, 138  
 Export Source ..... 111  
 ExportSource ..... 138  
 File Recipient ..... 113  
 FileRecipient ..... 138  
 ReportPane ..... 115, 138  
 ReportService ..... 138  
 SectionHeader ..... 116, 138  
 ReportService ..... 111  
 Converters  
 types ..... 87  
 customizations ..... 41
- D**
- Data binding ..... 81  
 add ..... 6  
 types ..... 83  
 Distribution ..... 57  
 documentation  
 related ..... iii
- E**
- Export Source ..... 111
- F**
- Field editor bindings ..... 90  
 File Recipient  
 component ..... 113
- G**
- Graphics  
 concepts ..... 49  
 principles ..... 49  
 tools ..... 49  
 graphics workflow ..... 3  
 Grid Editor view ..... 119  
 assign components ..... 121  
 Edit Column ..... 121  
 Edit Row ..... 121  
 workflow ..... 120  
 Grid Label Pane view ..... 117  
 Grid Table view ..... 116  
 gx  
 components ..... 132
- H**
- History Chart Builder  
 launch using a Popup binding ..... 16  
 Hx profiling visible property  
 u ..... 39
- I**
- Increment setpoint binding ..... 89
- K**
- Kiosk Profiles  
 Default Kiosk ..... 102  
 Handheld ..... 102  
 kitPx  
 components ..... 132  
 kitPxGraphics ..... 2, 69
- L**
- Login screen  
 create custom ..... 41
- M**
- Make Widget wizard  
 about ..... 102  
 Action widget ..... 34  
 bound label widget ..... 16  
 Chart widget ..... 20  
 component properties-based ..... 23  
 make widget ..... 8  
 palette kit-based ..... 25  
 Time Plot widget ..... 28  
 Workbench view-based ..... 31  
 Mobile  
 about ..... 122  
 NiagaraMobileWebApps ..... 122  
 Mobile Px app ..... 123
- N**
- Nav file  
 about ..... 104  
 add nodes ..... 44  
 building nav files ..... 42  
 create ..... 43  
 delete nodes ..... 44  
 delete using popup menu ..... 43  
 edit node hierarchy ..... 44  
 edit nodes ..... 45  
 elements  
 <nav> ..... 105  
 <node> ..... 105



- open a nav file..... 43
- rename ..... 43
- Nav File Editor
  - about..... 106
  - buttons ..... 109
  - controls ..... 106, 109
  - result tree..... 108
  - result tree pane ..... 106
  - source objects..... 107
  - source objects pane..... 106
- Nav file, editing ..... 43
- Notices
  - confidentiality.....2
  - copyright and patent.....2
  - trademark .....2
  
- O**
- ORD
  - absolute vs. relative..... 71
  - relativizing ..... 35
  
- P**
- Palette
  - bajaui ..... 76
  - kitPxGraphics ..... 69
  - Px Layers Palette ..... 62
  - Px Property ..... 60
- Pluggins
  - ComponentGrid..... 117
  - Grid Editor view ..... 119
- plugins..... 139
- Plugins
  - pxEditor..... 139
- Plugins module
  - report..... 139
- Popup Bindings..... 91
- Portability
  - examples..... 71
- Preface..... iii
- presentation
  - concepts ..... 49
  - principles..... 49
  - tools..... 49
- Presentation Xml (Px) ..... 1
- procedures.....3
- Profiles
  - about..... 93
  - Kiosk..... 101–102
  - Web
    - about ..... 94
    - Basic Hx ..... 99
    - Basic Touchscreen ..... 100
    - Basic Wb ..... 96
    - Default Hx ..... 97
    - Default Mobile..... 97
    - Default Touchscreen ..... 101
    - Default Wb ..... 94
    - Handheld Touchscreen..... 101
    - Hx ..... 39
    - Simple Admin Wb ..... 95
- property parameters
  - linking
  - unlinking..... 59
- Px
  - about..... 1
- Px Editor
  - Canvas..... 56
  - workflow..... 102
  - zoom..... 36–37
- Px file ..... 49
  - add comment box..... 4–5
  - embed history chart ..... 13
- Px files ..... 53
  - add comments box ..... 4
- Px files, shared ..... 54
- Px Layer
  - add a layer ..... 15
  - assign objects..... 15
  - remove layer ..... 15
  - rename layer..... 15
  - unassign objects..... 15
- Px Layers..... 14
  - about..... 61
  - Px Layers Palette ..... 62
- Px page
  - zoom..... 36–37
- Px properties..... 58
- Px Properties
  - concepts
    - property parameters ..... 59
    - using ..... 42
- Px Property palette..... 60
- Px target media
  - about types of ..... 92
  - HxPxMedia ..... 92
  - MobilePxMedia..... 92
  - ReportPxMedia ..... 92
  - WbPxMedia ..... 92
- Px View
  - create on component ..... 7
- Px View wizard..... 50
- Px viewer..... 55
- Px views ..... 49
- Px Views
  - visual styles ..... 42
- pxeditor
  - components..... 137
- PxEditor
  - about
    - canvas
      - side bar pane..... 55
    - zooming..... 63
- PxInclude ..... 11
  - embed Px file with ORD variables ..... 13
- PxInclude Widget..... 76

- concepts ..... 76
  - ORD variables..... 80
  - properties..... 78
  - using ..... 79
  - using with multiple ORD variables..... 39
  - using with single ORD variable..... 38
  - workflow..... 37
  - PxMedia type
    - choosing..... 51
- R**
- relativizing ORDs..... 91
  - report
    - components..... 137
  - report-Email Recipient
    - component..... 112
  - Reporting
    - about..... 109
    - EmailRecipient component..... 47
    - ExportSource..... 47
  - Reporting process
    - workflow..... 45
  - ReportingService
    - setup the ..... 46
  - ReportPane
    - component..... 115
  - ReportPxFile..... 46, 122
  - Reports
    - generating..... 45
    - layout data in ComponentGrid ..... 46
    - types of report components..... 110
  - ReportService..... 111
- S**
- SectionHeader
    - component..... 116
  - Setpoint binding
    - about..... 88
  - Slot properties
    - Px views as ..... 50
  - Spectrum bindings
    - about..... 88
  - Spectrum setpoint bindings..... 89
  - SVG
    - rendering ..... 63
  - SVG support..... 10, 12, 62
    - making AX-compatible SVGs..... 63
- T**
- Table bindings..... 90
  - tasks ..... 3
  - Third-party graphics ..... 71
- V**
- Value bindings
    - about..... 87
  - View Source XML..... 64
  - views..... 139
  - Views
    - ComponentGridEditor ..... 139–140
    - Grid Editor view..... 117, 119
    - Grid Table view..... 116
    - GridLabelPane..... 139
    - GridTable ..... 139
    - pxEditor..... 139
    - PxEditor..... 139
    - report..... 139–140
    - report module..... 139
    - reportGridLabelPane..... 139
- W**
- Widget
    - add binding ..... 6
    - adding
      - using Make New Widget wizard..... 8
    - animate property..... 9
    - commands ..... 68
    - create
      - Action..... 34
    - layout ..... 65
    - make
      - bound label widget..... 16
      - Chart ..... 20
      - component properties-based..... 23
      - palette kit-based module ..... 25
      - Time Plot..... 28
      - Workbench-view based..... 31
    - painting ..... 68
    - panes..... 66
    - Picture..... 12
    - properties..... 69
    - relationships..... 65
    - scalable..... 12
  - Widget sizing..... 58
  - Widget widget
    - add a weather report..... 8
  - Widgets
    - about..... 65
    - bajau ..... 65
    - kitPx..... 65
    - kitPxGraphics ..... 65
    - kitPxHvac ..... 65
    - Picture..... 10
    - PxInclude..... 76
  - Widgets, Picture ..... 62
- Z**
- Zoom

about..... 63  
capability ..... 63  
reset zoom..... 37  
zoom-in..... 36  
zoom-out..... 37  
Zooming ..... 36