

Technical Document

Niagara^{AX-3.6} Sedona Framework Networks Guide

November 10, 2011



Niagara^{AX} Sedona Framework Networks Guide

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 Visio are registered trademarks of Microsoft Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Mozilla and Firefox are registered trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, Niagara^{AX} Framework, and Sedona Framework are registered trademarks, and Workbench, WorkPlace^{AX}, and ^{AX}Supervisor, 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.

Copyright © 2011 Tridium, Inc.

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

CONTENTS

| | |
|--|----------------|
| Preface | v |
| About this document | v |
| Sedona Framework network FAQs | v |
| Sedona Framework terms | vi |
| Document change log | vii |
| Sedona Framework Network Driver Installation | 1-1 |
| Licensing and module requirements | 1-1 |
| Sedona Network Quick Start | 2-1 |
| Configure a SedonaNetwork | 2-1 |
| Add a SedonaNetwork | 2-1 |
| <i>Add a SedonaNetwork</i> | <i>2-1</i> |
| Add SedonaDevices | 2-2 |
| <i>Add SedonaDevices</i> | <i>2-2</i> |
| Configure a SedonaJen6lpNetwork | 2-3 |
| Add a SedonaJen6lpNetwork | 2-3 |
| <i>Add a SedonaJen6lpNetwork</i> | <i>2-3</i> |
| Configure key SedonaJen6lpNetwork properties | 2-4 |
| <i>Configuring key SedonaJen6lpNetwork properties</i> | <i>2-4</i> |
| Discover and add SedonaJen6lpDevices | 2-5 |
| <i>Adding discovered Jennic-based devices</i> | <i>2-5</i> |
| Rename SedonaJen6lpDevices | 2-7 |
| Service pin method | 2-7 |
| Match address map entries to recorded MAC addresses | 2-7 |
| Run the Manifest Manager to ensure kit manifests are loaded | 2-8 |
| <i>To run the Sedona Manifest Manager for the Sedona Framework network</i> | <i>2-9</i> |
| Create Sedona proxy points (and action points) | 2-10 |
| <i>To create Sedona proxy points and action points</i> | <i>2-10</i> |
| Configure to allow Sox tunneling | 2-13 |
| <i>Add the SoxTunnel service</i> | <i>2-13</i> |
| Niagara Sedona Framework Concepts | 3-1 |
| Understanding the different levels of data access | 3-1 |
| About Sedona Framework network types | 3-2 |
| Sedona Framework networks: quick comparison | 3-2 |
| Sedona Framework network component differences | 3-3 |
| Sedona device component differences | 3-4 |
| Similarities in Sedona Framework network types | 3-5 |
| About the SedonaNetwork component | 3-5 |
| SedonaNetwork properties | 3-6 |
| SedonaNetwork tuning policy notes | 3-6 |
| SedonaNetwork poll scheduler notes | 3-6 |

| | |
|--|-------------|
| <i>SedonaNetwork Comm Config</i> | 3-7 |
| <i>SedonaNetwork debug properties</i> | 3-7 |
| <i>Sedona Device Manager view</i> | 3-7 |
| <i>Adding SedonaDevice notes</i> | 3-7 |
| <i>Sedona Device Manager data columns</i> | 3-8 |
| <i>Manifest Manager view of SedonaNetwork</i> | 3-9 |
| About the SedonaDevice component | 3-10 |
| About the SedonaJen6lpNetwork component | 3-11 |
| About Jennic-based devices | 3-11 |
| Hibernating devices | 3-12 |
| About the Jennic RF network | 3-13 |
| Jennic-based device protocol stack | 3-13 |
| Jennic radio operation notes | 3-13 |
| SedonaJen6lpNetwork properties | 3-14 |
| SedonaJen6lpNetwork tuning policy notes | 3-14 |
| SedonaJen6lpNetwork poll scheduler notes | 3-16 |
| SedonaJen6lpNetwork Comm Config | 3-16 |
| SedonaJen6lpNetwork Chopan Server | 3-16 |
| SedonaJen6lpNetwork debug properties | 3-16 |
| SedonaJen6lpNetwork coordinator properties | 3-16 |
| SedonaJen6lpNetwork Pan Info properties | 3-18 |
| SedonaJen6lpNetwork Network Steady State Time property | 3-19 |
| SedonaJen6lpNetwork actions | 3-19 |
| Sedona Jen6lp Device Manager view | 3-20 |
| Jen6lp Device Manager Learn Mode notes | 3-20 |
| Jen6lp Device Manager data columns | 3-21 |
| Manifest Manager view of SedonaJen6lpNetwork | 3-21 |
| Pan Sheet view of SedonaJen6lpNetwork | 3-22 |
| About the SedonaJen6lpDevice component | 3-23 |
| SedonaJen6lpDevice properties | 3-23 |
| SedonaJen6lpDevice actions | 3-25 |
| About maintenance mode | 3-26 |
| About the Chopan Virtual gateway | 3-26 |
| Required configuration in the Jennic-based device | 3-26 |
| Walk-through using Chopan Virtual gateway | 3-27 |
| <i>Example Chopan point setup using Chopan Virtual gateway</i> | 3-27 |
| Additional Chopan usage topics | 3-32 |
| About the Sedona Point Manager | 3-33 |
| Unique things about the Sedona Point Manager | 3-33 |
| Sedona Point Manager "Discovered" notes | 3-34 |
| Add dialogs in Sedona Point Manager | 3-35 |
| <i>Sedona proxy point Add dialog</i> | 3-35 |
| <i>Sedona action point Add dialog</i> | 3-36 |
| Sedona Point Manager "Database" notes | 3-37 |
| <i>Database point table columns</i> | 3-37 |
| <i>Modifying the Database table</i> | 3-38 |
| About Sedona proxy points | 3-38 |
| Sedona and Niagara data types and null notes | 3-39 |
| Sedona property data types to default Sedona proxy point types | 3-39 |
| Niagara writable null status and Sedona | 3-39 |
| Booleans as Enums | 3-40 |
| About Sedona action points | 3-41 |

Sedona Framework Plugin Guides..... 4-1

| | |
|--|------------|
| Plugins in jen6lp module | 4-1 |
| Plugins in nsedona module | 4-1 |
| Plugins in pansheet module | 4-6 |
| Plugins in sedonanet module | 4-6 |

Sedona Framework Component Guides.....5-1

Components in jen6lp module 5-1**Components in nsedona module 5-2****Components in sedonanet module 5-4**

Sedona Framework Tools for WorkbenchA-1

Jennic Serial Port Tool A-2**New Jennic Wireless Adapter A-2****Sedona Installer A-3****Sedona Manifest Manager A-3**

PREFACE

Preface

This document explains NiagaraAX-3.6 and later integration of Sedona Framework devices, using Sedona Framework TXS (Sedona Framework 1.1).

Note: *Before AX-3.6, NiagaraAX integration of Sedona Framework 1.0 devices used a different architecture, including Niagara components sourced from different modules.*

This preface has the following sections:

- [About this document](#)
- [Sedona Framework network FAQs](#)
- [Sedona Framework terms](#)
- [Document change log](#)

About this document

As noted earlier in this [Preface](#), this document applies to NiagaraAX integration of Sedona Framework devices in AX-3.6 and later, and has the following main sections:

- [Sedona Framework Network Driver Installation](#)
Explains the NiagaraAX platform, software, and licensing requirements.
- [Sedona Network Quick Start](#)
Provides several step-by-step procedures for getting started with a Sedona Framework network (either SedonaNetwork or SedonaJen6lpNetwork), working online with a station.
- [Niagara Sedona Framework Concepts](#)
Provides concepts behind both types of Sedona Framework networks, including most components and views. Includes reference information on almost all properties and actions of components, from networks to devices to proxy points and action points.
- [Sedona Framework Plugin Guides](#)
Provides brief summaries of the various Sedona Framework-related views, with links back to the more detailed concepts sections. Entries are used in NiagaraAX Workbench context-sensitive help “On View”.
- [Sedona Framework Component Guides](#)
Provides brief summaries of the different Sedona Framework-related components, with links back to the more detailed concepts sections. Entries are used in NiagaraAX Workbench context-sensitive help “Guide On Target”.
- Appendix A - [Sedona Framework Tools for Workbench](#)
Provides overview information about various tools added to NiagaraAX Workbench when enabled for Sedona Framework TXS, along with links to other documents with more details.

Sedona Framework network FAQs

Below are some frequently asked questions (FAQs) about NiagaraAX integration of Sedona Framework devices.

Q: What is the difference between Sedona Framework TXS and Sedona Framework?

A: Sedona Framework TXS (Tridium eXtended Services) represents all of the value added, commercial components of the Sedona Framework that Tridium has created above and beyond the Open Source Sedona Framework release. Sedona Framework TXS 1.1 is based on the NiagaraAX-3.6 release and the Open Source Sedona 1.0.48 release.

Sedona Framework terms

The following is a list of terms and abbreviations used in this document when describing networks of Sedona Framework devices. For general NiagaraAX terms, see the Glossary in the *User Guide*. Note that this glossary may grow over time, or may else simply be eliminated.

6LoWPAN Acronym for IPv6 over Low power Wireless Personal Area Networks. It is an international open standard that enables using 802.15.4 and IP together. Sedona was created with 6LoWPAN networking capability in mind, reflected in its [DASP](#) and [Sox](#) protocols.

app The app in a Sedona Framework device is its application of Sedona components and services, including links between them, plus all configuration properties. Components and services are selected from [kits](#) installed in the device. In this way, an app is analogous to a station on a NiagaraAX platform. Using a Sox connection, Workbench is used to engineer an app and save it to the device's Flash memory. Sedona Sox Tools in Workbench include an "App Manager" view for getting/putting a device's app.

A Sedona Framework device may also have a "fallback app", residing in a different area of Flash memory. A fallback app is started via some local device-specific method, for example, pressing push-buttons in combination.

bundle Sedona Framework TXS software is distributed in a "bundle". A bundle is a special image (set of files) that is available from Niagara-Central (www.nigara-central.com). You use the **Sedona Installer** tool in Workbench to install a bundle. For complete details, refer to the *NiagaraAX Sedona Installer Guide*.

Chopan Or CHoPAN, for Compressed HTTP over Personal Area Networks. It is a Tridium proprietary protocol used in Sedona Framework TXS to support Jennic-based devices, particularly battery-powered devices. Chopan runs over UDP/IP and is a "session-less" protocol (unlike Sox), offering a number of advantages over [Sox](#) in certain applications. For related details, see "[About the Chopan Virtual gateway](#)" on page 3-26.

coordinator In a network of [Jennic-based](#) devices, the JACE station acts as the single "coordinator" node for all child nodes, using the Sedona Jennic option card installed in that JACE. The coordinator maintains info about its child nodes, each of which may provide routing functionality or be end devices. Properties of the station's SedonaJen6lpNetwork configure the coordinator's operating parameters. See "[SedonaJen6lpNetwork coordinator properties](#)" on page 3-16.

Alternatively, a special Sedona Jennic "USB stick" can be used as the coordinator with Sedona Framework Workbench (or Sedona Framework TXS-enabled Niagara Workbench) to support a Sox connection to a *single* Jennic-based device. No network functionality is possible. Setup of this USB coordinator is done using the "New Jennic Wireless Adapter" tool in Workbench. See the *Jennic Serial Tools Guide* for details.

DASP For Datagram Authenticated Session Protocol. This is the low-level, secure session-based protocol that [Sox](#) utilizes. DASP operates in networks that include 6LoWPAN and resource-limited devices. Sedona Framework network and device components in NiagaraAX provide debug properties that allow examining DASP and Sox messaging.

hibernating device Refers to a type of [Jennic-based](#) device that is typically powered by an onboard battery or batteries. Such a device invariably "hibernates" (sleeps) the majority of time, periodically "waking up" for short periods to execute routines and exchange data with other devices. Such devices require configuration using CHoPAN ([Chopan](#)). See "[Hibernating devices](#)" on page 3-12.

Currently, Sedona Framework support for hibernating devices is not widely available. However, the SedonaJen6lpNetwork driver in the NiagaraAX station is "ready" for such device support if this changes. Other sections of this document that mention hibernating devices also note this.

Jennic-based A Jennic-based device is the term used in Tridium tech docs for a wireless Sedona Framework device based on a Jennic micro-controller, with built-in 802.15.4 connectivity and 6LoWPAN stack support. Such devices are modeled as "SedonaJen6lpDevices" in the NiagaraAX station of a JACE controller (with an installed "Sedona Jennic" option card), under a SedonaJen6lpNetwork. See "[About Jennic-based devices](#)" on page 3-11.

JenNet The Jennic protocol that manages wireless 802.15.4 network formation and message routing, sitting above the 802.15.4 layer and below the [6LoWPAN](#) layer. JenNet provides a "self healing tree" network, versus a mesh network. In a SedonaJen6lpNetwork, the JACE station is always the top "coordinator" node of the network tree. A special diagnostic "Pan Sheet" view of the network provides JenNet data.

kit Sedona kits are the basic unit of modularity of Sedona software, encapsulating code, types, and metadata. A kit is analogous to a module on a NiagaraAX platform. The [app](#) in a device instantiates components and services contained in its installed kits. You must have the appropriate kits available on your Workbench platform to change a device's "core" software. Sedona Sox Tools in Workbench include a "Kit Manager" view to manage kits on a Sedona Framework device.

manifest Each kit has a corresponding manifest, with all metadata needed by tools like Workbench to Sox connect to a Sedona Framework device, and for a station to support Sedona proxy points in that device. Manifest files are compact XML files, named using a *kitName-checksum* convention similar to kit files, but with an `.xml` extension.

You use the Manifest Manager view of either Sedona Framework network type to manage kit manifests on the NiagaraAX host (JACE or Supervisor), or the Sedona Manifest Manager view in Workbench tools to manage manifests on your Workbench host. For related details in this document, see ["Run the Manifest Manager to ensure kit manifests are loaded"](#) on page 2-8. For complete details, refer to the *Sedona Manifest Manager - Engineering Notes* document.

PAN Personal Area Network, a generic term for a device network that is typically limited to a small area.

Sedona Framework device A Sedona Framework device runs a Sedona Framework [app](#) in a Sedona Framework VM (virtual machine), using installed Sedona Framework [kits](#), and is configured in Workbench using a [Sox](#) connection. Sedona Framework devices may vary in a number of ways, including device connectivity—for example Ethernet/IP, WiFi, or 802.15.4 (wireless PAN).

Sox Sox is the standard protocol used to communicate with Sedona Framework devices. It runs over UDP via the lower-level [DASP](#) protocol. Workbench always uses Sox to connect to (open) a Sedona Framework device. A Niagara station also uses Sox to discover and (typically) read and write to Sedona proxy points. However, if a Jennic-based device is configured with [Chopan](#), that comm type can be used for proxy point updates.

WPAN Wireless Personal Area Network, a [PAN](#) (personal area network) using a wireless technology.

Document change log

Updates (changes/additions) to this *Niagara^{AX} Sedona Framework Networks Guide* document are listed below.

- Publication: November 10, 2011
Initial document.

CHAPTER 1

Sedona Framework Network Driver Installation

There are currently two different NiagaraAX Sedona Framework network (or driver) types, depending on communications method between the NiagaraAX station and the Sedona Framework devices:

- **SedonaNetwork**
For Sedona Framework devices that provide an Ethernet port (or WiFi) that utilizes TCP/IP. This Sedona Framework driver is capable of running on any AX-3.6 or later platform, meaning any model JACE or a Supervisor.
- **SedonaJen6lpNetwork**
For wireless “Jennic-based” devices, which use 6LoWPAN over 802.15.4. This Sedona driver requires a JACE-2, -6, -7 or JACE-x02 XPR (with installed Sedona Jennic option card) as host platform, running AX-3.6 or later. This driver is licensed separately from the other (SedonaNetwork) one.

If needed, and if properly licensed, a JACE station can have *both* types of Sedona Framework networks.

Licensing and module requirements

Note: Before upgrading a JACE from a previous revision, or before installing software modules in a JACE, use the Workbench “Sedona Installer” tool to install the latest Sedona Framework TXS 1.1 bundle for Workbench, and restart Workbench. This ensures the latest Sedona Framework-related modules are available in Workbench for installation in remote JACE controllers. For details, see the NiagaraAX Sedona Installer Guide document.

That document also includes a complete summary of Sedona Framework licensing in NiagaraAX.

- The SedonaNetwork requires the NiagaraAX host to have the `sedonanet` and `nsedona` modules installed, and its license to have the `sedonanet` feature. Device and/or proxy point limits may exist (see below).
- The SedonaJen6lpNetwork requires the JACE host to have the `sedonanet`, `nsedona`, `jen6lp`, and `platJen6lp` modules installed, and its license to have the `jen6lp` feature. Device and/or proxy point limits may exist (see below).
- For either network, the maximum number of devices supported is determined in the host’s license, in the `device.limit` attribute of its network feature (`sedonanet` or `jen6lp`). If more than the specified number of device components are added to the station, they remain in fault until deleted, or until the license is upgraded for more devices.
However, note that the license feature may also limit the total number of proxy points, in the `point.limit` attribute. This point limit applies across all child devices in the network, and may make a device limit irrelevant. For example, consider if a JACE’s license has a `sedonanet` (feature) device limit of 20 and a point limit of 200. If under its SedonaNetwork, 20 SedonaDevices were modeled in the station with 10 proxy points each, both limits would be reached with good results. However, if each device was modeled in the station with 15 proxy points, any proxy points added after the 200 point limit would remain in fault.
- If Sox tunneling (through either network type) is needed, the NiagaraAX host requires a license that has the `tunneling` feature with the attribute `sox="true"`.

Apart from installing the 3.6.*nnn* version of the Niagara distribution in the JACE, make sure to install the modules noted above, plus any others needed for other drivers or features. Upgrade any modules shown as “out of date”. For details, see “Software Manager” in the *Platform Guide*.

CHAPTER 2

Sedona Network Quick Start

This section provides a collection of procedures to get started with either NiagaraAX Sedona Framework driver in typical online scenarios. Like other NiagaraAX drivers, you can do most configuration from special “manager” views and property sheets using Workbench. These are the main subsections:

- [Configure a SedonaNetwork](#)
 - [Add a SedonaNetwork](#)
 - [Add SedonaDevices](#)
- [Configure a SedonaJen6lpNetwork](#)
 - [Add a SedonaJen6lpNetwork](#)
 - [Configure key SedonaJen6lpNetwork properties](#)
 - [Discover and add SedonaJen6lpDevices](#)
 - [Rename SedonaJen6lpDevices](#)
- [Run the Manifest Manager to ensure kit manifests are loaded](#)
- [Create Sedona proxy points \(and action points\)](#)
- [Configure to allow Sox tunneling](#)

Note: A separate engineering notes document, “Using the Sedona Jennic option card,” also provides basic steps to get started with the **SedonaJen6lpNetwork** driver for wireless Jennic-based devices, and includes additional overview and background information.

Configure a SedonaNetwork

The SedonaNetwork (for Ethernet and WiFi Sedona Framework devices) is the simplest of the two types of Sedona Framework networks. To configure a SedonaNetwork, perform the following main tasks:

- [Add a SedonaNetwork](#)
- [Add SedonaDevices](#)

Add a SedonaNetwork

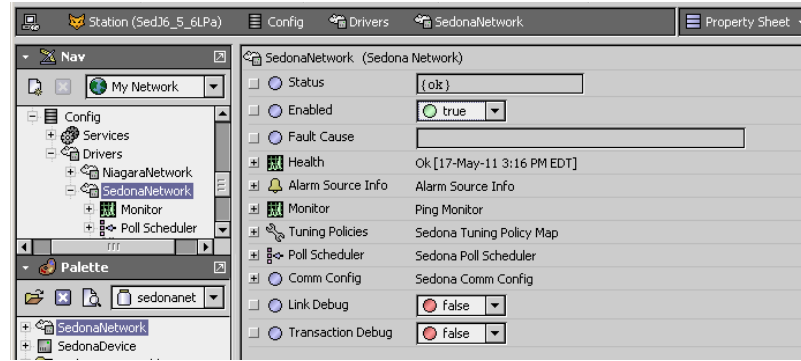
Note: The host requires certain modules and licensing. See “[Licensing and module requirements](#)” on page 1-1.

Add a SedonaNetwork

With the station open in Workbench:

- Step 1 In the Nav tree, expand the station’s **Config** node to reveal the **Drivers** container.
- Step 2 Open the **sedonanet** palette in your Workbench palette side bar (see “Using the palette side bar” in the *User Guide* for general details).
- Step 3 From the **sedonanet** palette, *drag* (or copy and paste) a **SedonaNetwork** into the station’s **Drivers** folder. In the popup **Name** dialog, you can rename the network—or, simply use the default name. A SedonaNetwork named “SedonaNetwork” (or whatever else), is under your Drivers folder ([Figure 2-1](#)).

Figure 2-1 SedonaNetwork property sheet



Leave properties at defaults for now. For details, see [“About the SedonaNetwork component”](#) on page 3-5.

Add SedonaDevices

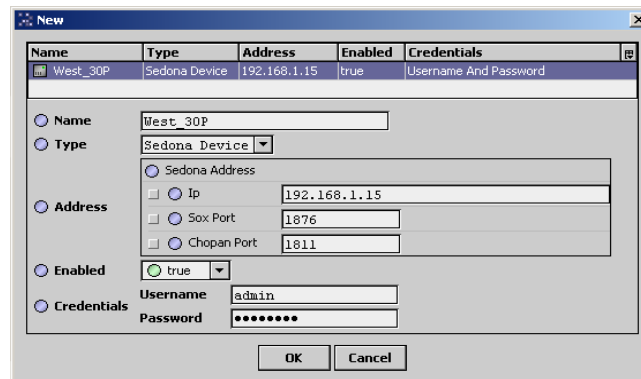
SedonaDevices represent Ethernet/IP or WiFi-equipped Sedona Framework devices.

Note: You need to know the IP addresses of the Sedona Framework devices. The IP address for each device should each be unique, and reachable by the Niagara station.

Add SedonaDevices

- Step 1 Double-click the SedonaNetwork for its **Sedona Device Manager** view.
- Step 2 In the Sedona Device Manager, click the **New** button to add a **New** Sedona Device.
- (Optional) Use the **New Folder** feature to make one or more device folders, each with its own device manager view—a common method to “group” devices. Then double-click a folder and click **New**.
- Click **OK** for the **New** dialog for adding a device (Figure 2-2).

Figure 2-2 New dialog for adding a SedonaDevice



- In the **Name** field, you typically enter another name instead of “SedonaDevice” (may be changed later at any time). This is a Niagara name—it does not need to match any Sedona Framework app or component name.
- In the **Sedona Address** area:
 - In the **Ip** field, enter the known IP address of the Sedona Framework device. This may be either an IPv4 address, (e.g. 192.168.1.4) or and IPv6 address (e.g. FE80:0000:0000:0000:0215:8D00:000E:41f7)
 - Leave **Sox Port** at default 1876 (unless the device’s Sedona Framework app uses a different Sox port).
 - *Disregard Chopan Port* — currently, Chopan applies to SedonaJen6lpDevices only.
- Leave **Enabled** at true.
- In the **Credentials** area, enter Sedona credentials for a Sox connection.
 - Typically you leave **Username** at default admin (for this standard user).
 - In **Password** field, enter the required password.
- Click **OK** to add the SedonaDevice to the network.

If successful, you should see the device with “ok” Health in the Sedona Device Manager (Figure 2-3).

Figure 2-3 Newly-added SedonaDevices

The screenshot shows the 'Sedona Device Manager' window. It has a menu bar with 'Config', 'Drivers', 'SedonaNetwork', and 'Sedona Device Manager'. Below the menu is a 'Database' section with a table titled '2 objects'. The table has columns: Name, Type, Exts, Address, Status, Health, and Fault. There are two rows of data:

| Name | Type | Exts | Address | Status | Health | Fault |
|----------|---------------|------|--------------|--------|----------------------------|-------|
| East_30P | Sedona Device | | 192.168.1.10 | {ok} | Ok [18-May-11 9:13 AM EDT] | |
| West_30P | Sedona Device | | 192.168.1.15 | {ok} | Ok [18-May-11 9:10 AM EDT] | |

At the bottom of the window are buttons: 'New Folder', 'New', 'Edit', 'Discover', 'Cancel', and 'Add'.

Step 3 Repeat [Step 2](#) to add other Ethernet and/or WiFi Sedona Framework devices. As with other drivers, you can add multiple devices in a single **New** dialog—in this case, click on the row for each device to edit properties like Name, Ip, etc.

Other device component properties and containers are available on each device's property sheet. For related details, see [“About the SedonaDevice component”](#) on page 3-10.

Each device has a points extension, with a default **Sedona Point Manger** view. However, before adding proxy points you should make sure the station's host has the necessary Sedona manifest files.

For quick start procedures, see:

- [“Run the Manifest Manager to ensure kit manifests are loaded”](#) on page 2-8.
- [“Create Sedona proxy points \(and action points\)”](#) on page 2-10.

Configure a SedonaJen6lpNetwork

The SedonaJen6lpNetwork is an “extension” or “superset” of a SedonaNetwork, having all its features—plus additional ones to support 802.15.4 wireless (Jennic) operation, and other items for support of possible battery-powered (hibernating) devices.

To configure a SedonaJen6lpNetwork, perform the following main tasks:

- [Add a SedonaJen6lpNetwork](#)
- [Configure key SedonaJen6lpNetwork properties](#)
- [Discover and add SedonaJen6lpDevices](#)
- [Add SedonaDevices](#)

Add a SedonaJen6lpNetwork

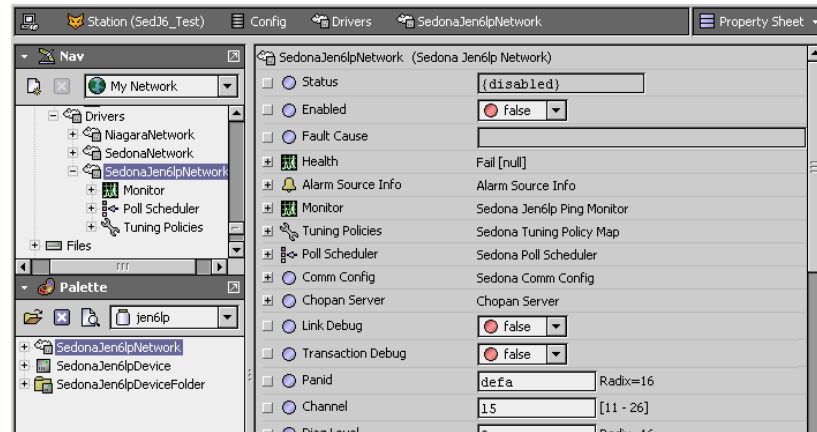
Note: The JACE host controller requires an installed Sedona Jennic option card, as well as certain modules and licensing. See [“Licensing and module requirements”](#) on page 1-1.

Add a SedonaJen6lpNetwork

With the station open in Workbench:

- Step 1 In the Nav tree, expand the station's **Config** node to reveal the **Drivers** container.
- Step 2 Open the **jen6lp** palette in your Workbench palette side bar (see “Using the palette side bar” in the *User Guide* for general details).
- Step 3 From the **jen6lp** palette, *drag* (or copy and paste) a **SedonaJen6lpNetwork** into the station's **Drivers** folder. In the popup **Name** dialog, you can rename the network—or, simply use the default. A SedonaJen6lpNetwork named “SedonaJen6lpNetwork” (or whatever you named it), is under your Drivers folder.
- Step 4 As copied from the palette, the SedonaJen6lpNetwork is disabled, and requires entered values in several configuration properties before it can be operational ([Figure 2-4](#)).

Figure 2-4 SedonaJen6lpNetwork property sheet for newly-added network



See “Configure key SedonaJen6lpNetwork properties” on page 2-4.

Configure key SedonaJen6lpNetwork properties

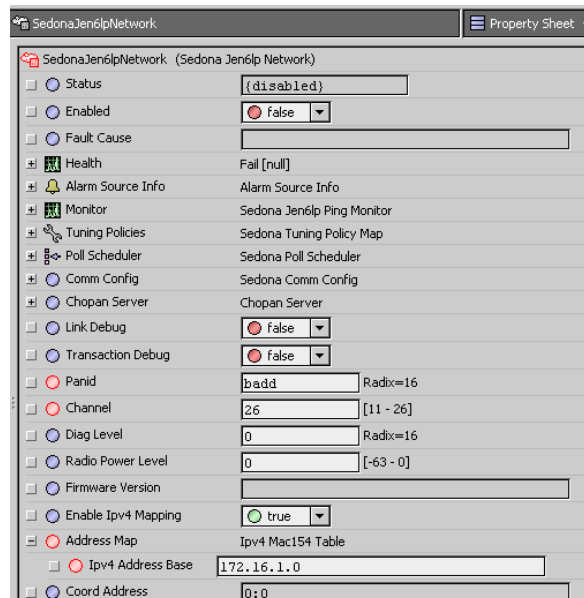
Do this after adding the SedonaJen6lpNetwork (see “Add a SedonaJen6lpNetwork” on page 2-3).

Note: You must know the Jennic PANID and channel map range used by the wireless Jennic-based devices, in order to configure the Sedona Jennic option card in the JACE to operate as the coordinator for them.

Configuring key SedonaJen6lpNetwork properties

Step 1 Open the property sheet for the **SedonaJen6lpNetwork**, if not already open

Figure 2-5 Set desired Panid, Channel, and Address Map for Sedona Jennic option card in JACE



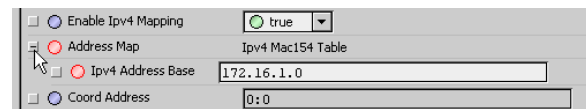
Step 2 Set the **Panid** and **Channel** used by the wireless network of Jennic-based devices. A few properties of this network are described as follows:

- **Enabled**
Initially false (disabled). After configuring and saving other properties in remaining steps, you change it to true and save, to initiate all changes.
- **Panid**
Jennic PAN ID (Personal Area Network identifier), in hexadecimal, range 0x0000 to 0xFFFF for the network. The JACE station acts as the coordinator for this network. Panid should match the PANID in use by installed devices. The default Panid is: defa
- **Channel**
RF 2.4GHz channel number to use, from 11 to 26. This channel must be included in the “channel map” of installed devices.

- **Address Map**
Container to specify the IPv4 address network “base” as well as hold entries for discovered nodes.
See the next step.

Step 3 Expand the **Address Map**.

Figure 2-6 Expanded Address Map



In the **IPv4 Address Base** field, either accept the default address base network (192.168.1.0), or enter another private IPv4 subnet for the JACE to map discovered wireless Jennic-based devices.

Note: The IPv4 address base must be on a different subnet than the subnet used by the JACE for normal IP communications. For further details, see the **Address Map** description in the section “SedonaJen6lpNetwork coordinator properties” on page 3-16.

The **IPv4 Address Base** subnet must fall within the Class A, B, or C address range, as follows:

- 10.0.0.0 to 10.255.255.0 (Class A)
- 172.16.0.0 to 172.31.255.0 (Class B)
- 192.168.0.0 to 192.168.255.0 (Class C)

As new wireless devices are discovered, they are assigned the next available IP address. The first address ($n.n.n.0$) is reserved for the coordinator.

For example, with the base address at default (192.168.1.0), the coordinator is 192.168.1.0, and the first device detected will be assigned 192.168.1.1, the next device 192.168.1.2, and so on. These devices automatically appear as dynamic entries under the network’s Address Map.

Step 4 Save all changes made in the SedonaJen6lpNetwork property sheet.

Step 5 Make sure the **Enable IPv4 Mapping** property is true, and set the network’s **Enable** property to true, and Save again.

The status of the network should change from disabled to ok, and the “Coord Address” should change from 0:0 to an actual address (for example: 158d00:9b50b). Under the **Address Map** in the property sheet, entries should begin to populate under the IPv4 Address Base. Within a minute or two, all address map entries should be complete.

Note: In the future, after changing any of the following network properties, you must disable then re-enable the SedonaJen6lpNetwork (or else restart the station) for them to be effective:

- *Panid*
- *Channel*
- *Radio Power Level*

For more details on *key properties* of the network, see “SedonaJen6lpNetwork coordinator properties” on page 3-16. For reference information on all network properties and actions, see “SedonaJen6lpNetwork properties” on page 3-14 and “SedonaJen6lpNetwork actions” on page 3-19.

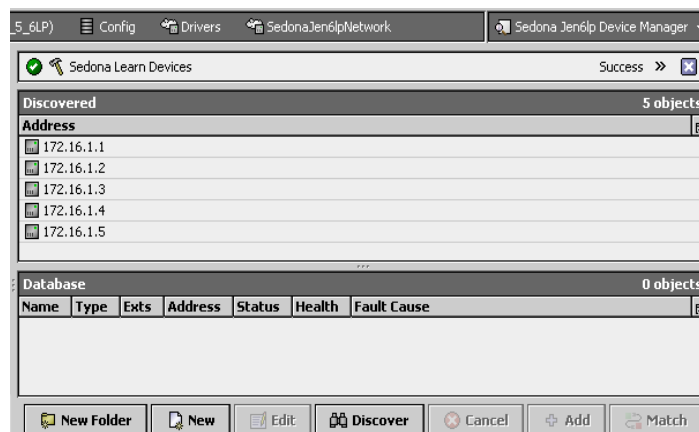
Discover and add SedonaJen6lpDevices

Do this after configuring key SedonaJen6lpNetwork properties (see “Configure key SedonaJen6lpNetwork properties” on page 2-4). SedonaJen6lpDevices represent wireless Jennic-based devices. For background details, see “About Jennic-based devices” on page 3-11.

Adding discovered Jennic-based devices

Step 1 Double-click the SedonaJen6lpNetwork for its default Device Manager view, and click the **Discover** button. A “Sedona Learn Devices” job quickly runs, and shows discovered devices in the top pane.

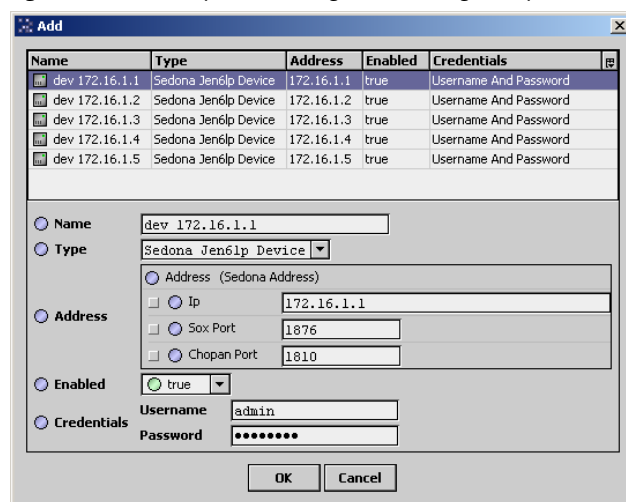
Figure 2-7 Example discovered wireless Jennic-based devices



(Optional) Use the **New Folder** feature to make one or more device folders, each with its own device manager view—a common method to “group” devices. Then double-click a folder and do a discover.

Step 2 Click to select one or more devices, then click the **Add** button. [Figure 2-8](#) shows the **Add** dialog.

Figure 2-8 Example Add dialog when adding multiple devices, all with default entries



Often properties can be left at defaults, including (initially) Name. After adding devices to the database you can rename devices with more meaningful text. See [“Rename SedonaJen6lpDevices”](#) on page 2-7.

Step 3 Click **OK** to add devices to the database.

Figure 2-9 Example added Jennic-based devices with default names

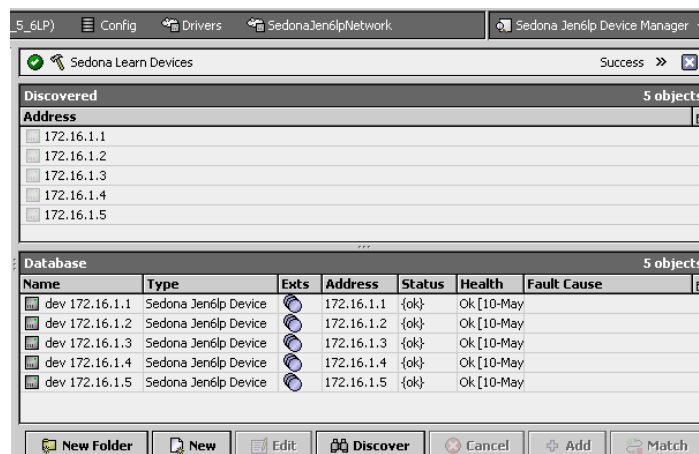


Figure 2-9 shows all discovered devices added to the station database using default names. Typically, now you rename all the SedonaJen6lpDevice components. See “Rename SedonaJen6lpDevices” on page 2-7.

Other device component properties and slots are available on each device’s property sheet. For related details, see “About the SedonaJen6lpDevice component” on page 3-23.

Each device has a points extension, with a default **Sedona Point Manger** view. However, before adding proxy points you should make sure the station’s host has the necessary Sedona manifest files.

For quick start procedures, see:

- “Run the Manifest Manager to ensure kit manifests are loaded” on page 2-8.
- “Create Sedona proxy points (and action points)” on page 2-10.

Rename SedonaJen6lpDevices

Do this after adding discovered devices (see “Discover and add SedonaJen6lpDevices” on page 2-5).

Rename each device component according to its purpose and location. Note that wireless Jennic-based devices are initially mapped into the network’s specified IPv4 Address Base (subnet) in an undefined order—e.g. you cannot “pre-specify” in the Sedona Framework app of a device for it to be “device 1” or “device 2” on a network.

To force an order, you *could* power on only *one* device at a time, add it in the station, and then rename the device component as known appropriate. Apart from that, there are a couple of possible ways to make the association between an added Jen6lp device component and a specific Jennic-based device:

- [Service pin method](#)
- [Match address map entries to recorded MAC addresses](#)

Note: *It is recommended that you also set the “Device Type” property for every added SedonaJen6lpDevice, working from the device’s property sheet. Change from “Unknown” to either: “Router” or “End Device”, and then **Save**. Only devices known to be incapable of router functionality should be set to “End Device”. For related details, see “SedonaJen6lpDevice properties” on page 3-23.*

Service pin method

If the device’s Sedona Framework app supports it, a “service pin” can be invoked from the physical device. For example, this may be possible from a device by pressing a pushbutton. This can be useful to make a positive identification from that device.

For related details, see “Service Pin support” in the *Sedona Framework Chopan Usage* document.

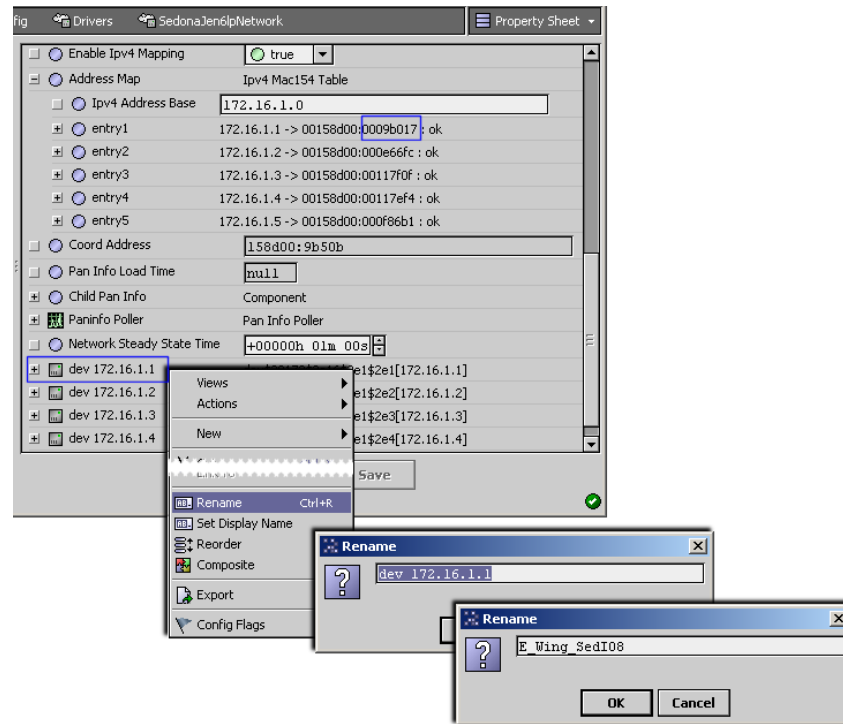
Match address map entries to recorded MAC addresses

This method always works, but it requires each device to have its unique MAC address on a printed label. It also requires careful note taking by the installer.

As you physically install the Jennic-based devices, make careful note of each one’s unique MAC address, recording the last four bytes (eight characters) of its 64-bit address. For example, if you install a device with MAC address 00158d00:00117f0f, record it as 00117f0f in a list.

Compare your recorded list of devices against the SedonaJen6lpNetwork’s “Address Map” entries (in the network’s property sheet, expand the Address Map and note the mapped IPV4 addresses to device MAC addresses). See [Figure 2-10](#).

Figure 2-10 MAC address of Address Map entries used against recorded list, to help device renaming



As shown in Figure 2-10, in the network's property sheet, below the **Address Map** area, you can right-click on each child device component, and rename it according to your notes.

Figure 2-11 Five example Jen6lp devices added and renamed in station

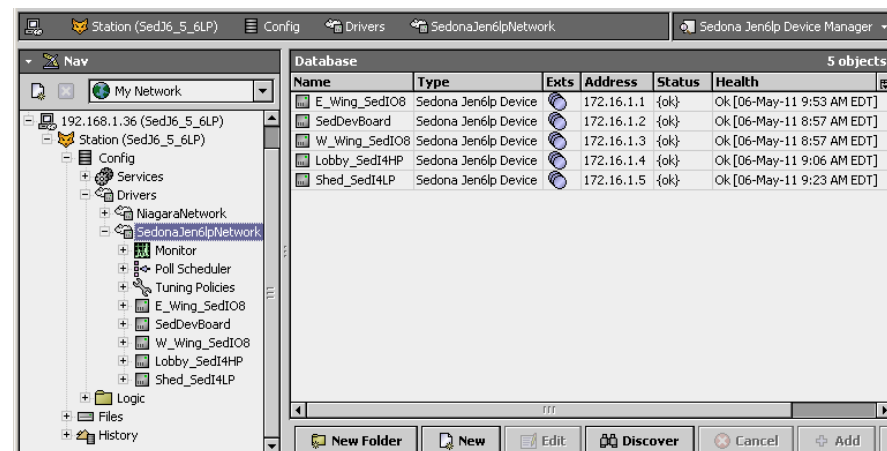


Figure 2-11 above shows five Jen6lp devices added and renamed in the station.

Run the Manifest Manager to ensure kit manifests are loaded

Note: Complete details on Sedona manifests and using the **Manifest Manager** are in the Sedona Framework Manifest Manager - Engineering Notes document.

To support Sedona proxy points, the NiagaraAX host (typically JACE) running the station with any Sedona Framework network requires all the Sedona “manifest” files for all the kits in each type of networked Sedona Framework device. These are the *same* manifest files required on your NiagaraAX Workbench host to support a Sox connection (or a *tunneled* Sox connection) to these devices.

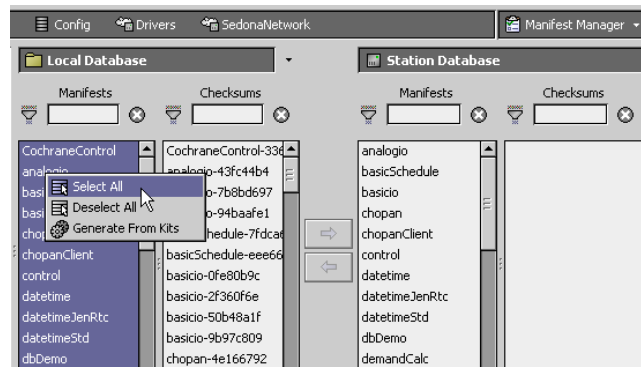
Note: The following quick start procedure assumes you already have all the necessary kit manifest files on your Sedona-enabled NiagaraAX Workbench host, located in the “manifests” subfolder of your working “sedona.home” (see your !lib/system.properties file for the sedona.home entry). For the purposes of this procedure, consider this source collection of manifest files your “Local Database”.

To run the Sedona Manifest Manager for the Sedona Framework network

This procedure applies to either Sedona Framework network type (SedonaNetwork or SedonaJen6lpNetwork), and results in the identical view and selections.

- Step 1 With the station opened, right-click the **SedonaNetwork** or **SedonaJen6lpNetwork** and select: **Views > Manifest Manager**. This brings up the **Manifest Manager**.
The *left side* of the view is your “**Local Database**”; the *right side* is the “**Station Database**”.
- Step 2 Right-click in the (far left) **Manifests** column of your **Local Database** and choose **Select All**.

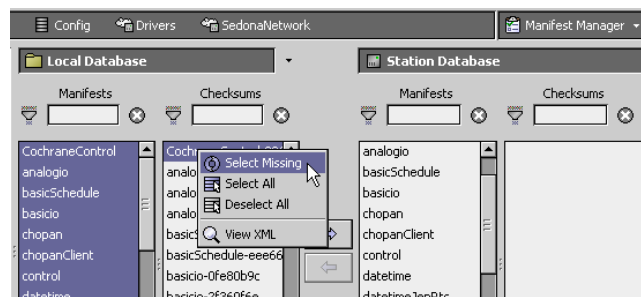
Figure 2-12 Right-click in Local Database's Manifests and choose "Select All"



All kits in the Manifests column become highlighted, as shown in Figure 2-12.

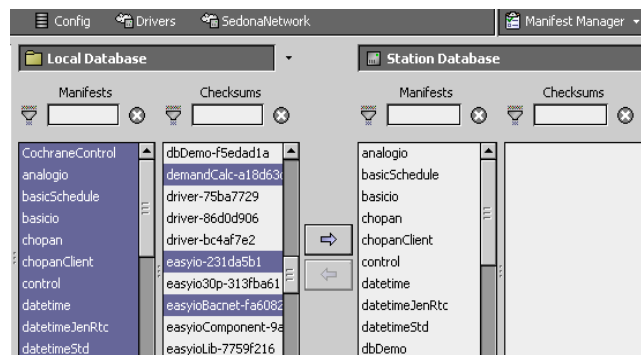
- Step 3 Right-click in the **Checksums** column of your **Local Database**, and choose **Select Missing**.


Figure 2-13 Right-click in Local Database's Checksums and choose "Select Missing"



This typically results in one or more **Local Database** “Checksums” entries becoming highlighted, which you can see by scrolling down that column, as shown in Figure 2-14 below. In some cases it may result in *all entries selected* (usually this is OK, as these files are small and do not use much space).

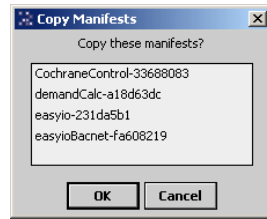
Figure 2-14 Manifests in the Local Database but missing in the Station Database are highlighted



The “Local Database” to “Station Database” manifest copy button  also becomes enabled.

- Step 4 Click this copy button to copy the manifests to the station's host.
This produces a confirmation dialog listing the manifest files to be copied (Figure 2-18)

Figure 2-15 Example confirmation dialog for manifest copy



- Step 5 Click **OK** to copy the manifest files to the station's host.
The manifests are transferred and are now immediately available in the hosting station to support Sedona proxy point operations, such as online point discovery.

Create Sedona proxy points (and action points)

Note: Before creating Sedona proxy points, you should make sure the station host (typically IACE) has the necessary Sedona kit “manifest” files for all the Sedona Framework devices. Otherwise, errors will occur when trying to discover points. See [“Run the Manifest Manager to ensure kit manifests are loaded”](#) on page 2-8.

As with device objects in other drivers, each Sedona Framework device (whether SedonaDevice or SedonaJen6lpDevice) has a Points extension that serves as the container for proxy points. The default view for any Points extension is the Point Manager (in this case, the **Sedona Point Manager**). You use it to discover and add Sedona proxy points and action points under any Sedona device component.

Note: The **Sedona Point Manager** differs from other point managers in that actions on Sedona components are separately (and individually) modeled in the station as “Sedona action points”, different from Sedona proxy points (standard Niagara control components with a Sedona proxyExt).

You create both point types, that is proxy points and action points, from discovered slots of Sedona components in a device using the Sedona Point Manager.

To create Sedona proxy points and action points

Once a Sedona device is added to the network (SedonaNetwork or SedonaJen6lpNetwork), you can discover the Sedona component slots in its app to select for modeling in the NiagaraAX station. Use the following procedure:



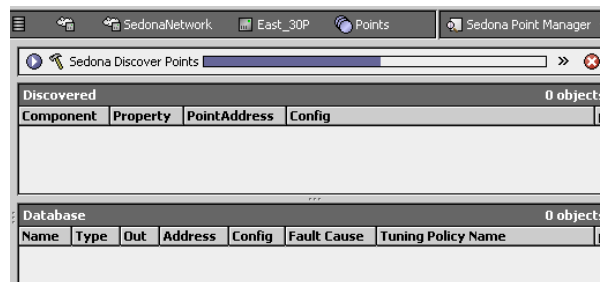
- Step 1 In the **Sedona Device Manager**, in the **Exts** column, double-click the **Points** icon  in the row representing the device you wish to explore.
This brings up the **Sedona Point Manager**.
- Step 2 Click  **Discover** to initiate point discovery using Sox connectivity. The view goes to a split-pane (learn mode).
(Optional) Use the **New Folder** feature to make one or more point folders, each with its own point manager view—a common method to “group” proxy points. Then double-click a folder and do a discover.
A Discover launches a “Sedona Discover Points” job, with progress bar near the top of the view.

Figure 2-16 Sedona Discover Points job in progress










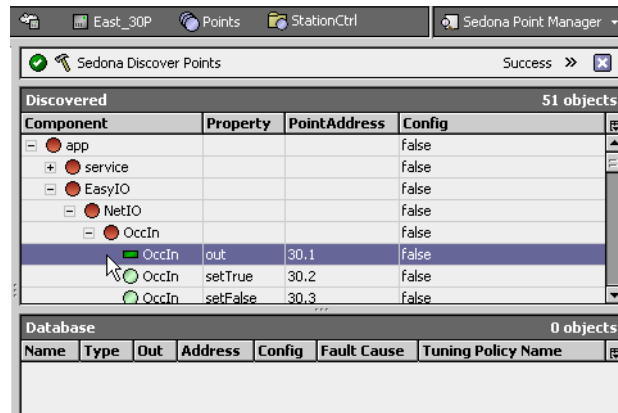
- The job ends with a single  **app** component in the Discovered pane, which you click  to expand.
- Step 3 Click to select the data items you wish to model in NiagaraAX. Folders and components appear as red  orbs, which you expand to see selectable slots—either properties ( ,  , ) or actions ().

Figure 2-17 Selecting Boolean property in Discovered pane



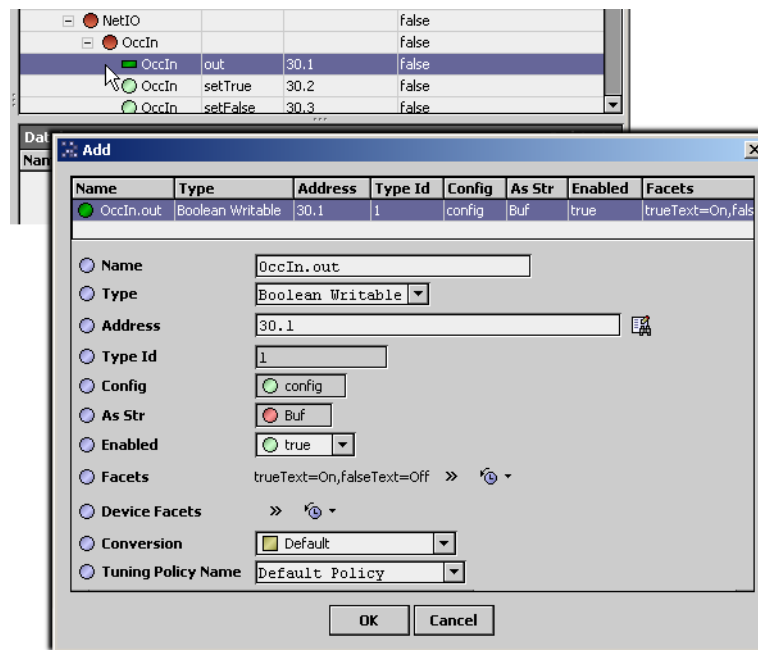
You can map selected items in the station in a number of ways:

- Drag from the Discovered pane to Database pane (brings up an **Add** dialog).
- Double-click an item in the Discovered pane (also brings up an **Add** dialog).
- Click to select in Discovered, then press “a”. (“Quick Add”, meaning *no* **Add** dialog).

This works the same as in other drivers’ Point Manager views (however, note if adding multiple items using the last method, all must *either* be properties *or* actions—but *not a combination of both*.)

- Step 4 When the **Add** dialog for a *property* appears, you can edit the configuration of that proxy point’s SedonaProxyExt before it is added in the Niagara station. Initial property values are determined by Niagara, based upon the property’s data type and whether config or runtime. See [Figure 2-18](#).

Figure 2-18 Example Add dialog for selected property



The following entries are in the **Add** dialog (and **Edit** dialog) for a Sedona component property:

- **Name** defaults to *SedonaComponentName.propertyName* for example, *OccIn.out* or *AvgTemp.in*. Where the *SedonaComponentName* is always 7 characters maximum. This is the Niagara point name only—change if needed (does *not* affect Sedona component).
- **Type** is the Niagara control point type to use for the proxy point. For most Sedona config types, the default selection is a “writable” control point; whereas runtime types are typically “read-only” points. Float and double data types are Numeric points; Integer and long data types default to Enum points. Boolean data types default to Boolean points; and string types are String points. For related details, see [“Sedona property data types to default Sedona proxy point types”](#) on page 3-39.
Note: Unlike other editable entries in the **Add** dialog, you cannot edit *Type* later in the **Edit** dialog.
- **Address** is the numerical Sedona address (compID.slotID) of the property slot. It is recommended

you do not manually edit this.

- **Type Id** is the numerical type ID of the data.
- **Config** reflects whether the value is stored in the device's non-volatile Flash memory (config) or not (runtime).
- **As Str** applies only if the target property is type `Sys::Buf`, and determines if Niagara attempts to interpret the byte array as a Sedona str(ing), or just leave it as a byte array.
- **Enabled** is to enable (true, the default) or disable the proxyExt. If set to disabled (false), the point's out value has a disabled status.
- **Facets** lets you set the proxy point's facets, where some default facets may exist (e.g., if Boolean, "true" and "false"), or facets may be "none". Typically, except for string data, you assign facets.
- **Device Facets** reflects native facets used in the device, if any.
- **Conversion** specifies the conversion to use between the "read value" (in Device Facets) and the parent point's facets, where "Default" is typically used.
- **Tuning Policy Name** specifies which of the available TuningPolicies in the network should be used to read data from and (if applicable) write data to the property. The "Default Policy" is the default selection—if other network tuning policies have been created (recommended), they are selectable in the drop-down control. For related details, see:
 - "SedonaNetwork tuning policy notes" on page 3-6
 - "SedonaJen6lpNetwork tuning policy notes" on page 3-14

Step 5 When you have Sedona proxy point(s) configured properly for your usage, click **OK**.

The proxy points are added to the station, and appear listed in the Database pane. Within a few seconds, the points will poll for current values.


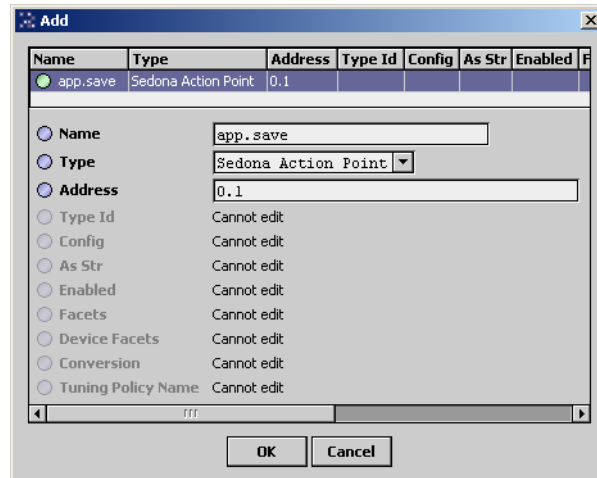

Note: If you selected one or more action slots () instead of properties, the Add dialog is completely different, as shown in [Figure 2-19](#).

Figure 2-19 Example Add dialog for action slot on a Sedona component



| Name | Type | Address | Type Id | Config | As Str | Enabled | F |
|--|---------------------|---------|---------|--------|--------|---------|---|
|  app.save | Sedona Action Point | 0.1 | | | | | |

☒ Name: app.save
☐ Type: Sedona Action Point
☐ Address: 0.1
☐ Type Id: Cannot edit
☐ Config: Cannot edit
☐ As Str: Cannot edit
☐ Enabled: Cannot edit
☐ Facets: Cannot edit
☐ Device Facets: Cannot edit
☐ Conversion: Cannot edit
☐ Tuning Policy Name: Cannot edit

OK Cancel

In this case, there are only three entry fields, as follows:

- **Name** defaults to `SedonaComponentName.actionName` for example, `app.save` or `UnochSP.set`. Where the `SedonaComponentName` is always 7 characters maximum. This is the Niagara point name only—change if needed (does not affect Sedona component).
- **Type** is always **Sedona Action Point**.
- **Address** is the numerical Sedona address (compID.slotID) of the action slot. It is recommended you do not manually edit this.

For more details on Sedona points in NiagaraAX, see the following sections

- "About the Sedona Point Manager" on page 3-33
- "About Sedona proxy points" on page 3-38
- "About Sedona action points" on page 3-41

Configure to allow Sox tunneling

Typically you configure a station with any Sedona Framework network to support “Sox tunneling”. This allows (full) Workbench users to open Sox connections to networked Sedona Framework devices, by “tunneling” through the station. However, if the station host is not licensed for Sox tunneling, you can skip this section. See [“Licensing and module requirements”](#) on page 1-1 for related details.

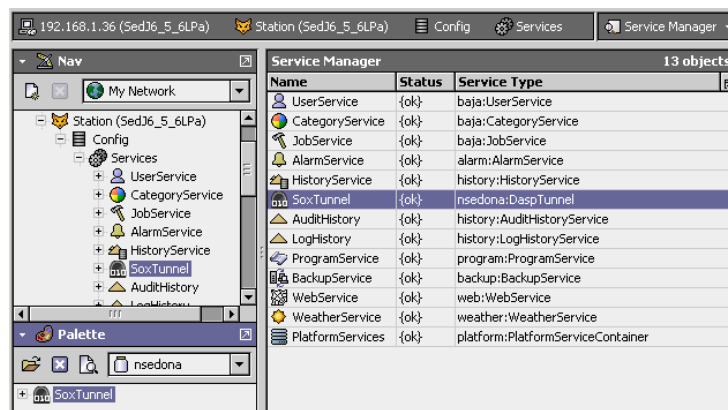
Note: Complete Sox tunneling details are in the Sedona Framework Sox Tunneling - Engineering Notes document. This procedure provides only a few simple steps to get started, and one connection method.

Add the SoxTunnel service

With the station open in Workbench:

- Step 1 In the Nav tree, expand the station’s **Config** node to reveal the **Services** container.
If a **SoxTunnel** is already there, skip ahead to [Step 4](#).
- Step 2 Open the **nsedona** palette in your Workbench palette side bar (see “Using the palette side bar” in the *User Guide* for general details).
- Step 3 From the **nsedona** palette, **drag** (or copy and paste) a **SoxTunnel** into the station’s **Services** container. In the popup **Name** dialog, you can rename the service—or, simply use the default name. The service named “SoxTunnel” (or whatever else), is under your Services container ([Figure 2-20](#)).

Figure 2-20 SoxTunnel service copied from nsedona palette



- Step 4 Providing the host is properly licensed for Sox tunneling, the service’s status should be “ok”.
Typically, no properties require adjustment, unless the target networked Sedona Framework devices are not using the default Sox port of 1876. In that case only, set the **Port** property of the SoxTunnel to match, and **Save**.
 - Step 5 To test the SoxTunnel service, in Workbench:
 1. Expand the SedonaNetwork or SedonaJen6lpNetwork to reveal child Sedona devices.
- Note:** If selecting a “hibernating” type device, you must first issue a “maintenance mode request” (action) on its SedonaJen6lpDevice, and wait for the popup dialog in Workbench before continuing. However, note that currently Sedona Framework support for hibernating devices is not widely available. For related details see [“About maintenance mode”](#) on page 3-26.
2. Right-click a Sedona device, and select **Open Sox - Tunnel Session**. An Authentication dialog appears, as shown below.

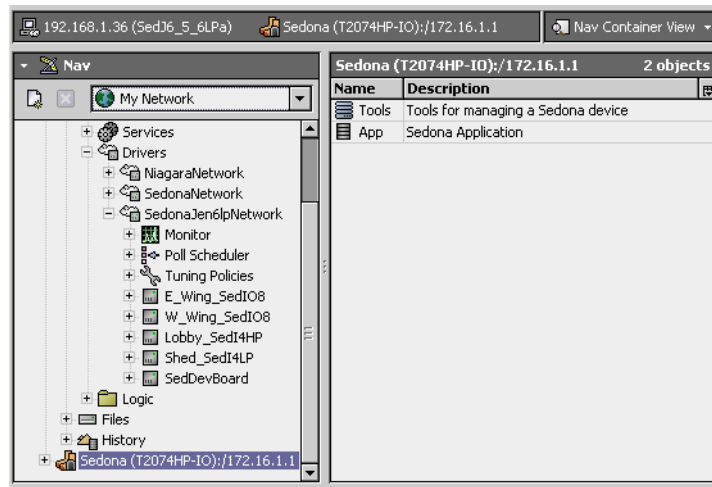
Figure 2-21 Authentication dialog for Sox connection



- Step 6 Enter the Username and password of a User in the Sedona Framework app's UserService (for example: the frozen admin user), and click **OK**. A Sox connection is made tunneling through the running station.
- The view space shows the Sedona device's **Nav Container View**, with **Tools** and **App** nodes, as shown in [Figure 2-22](#).

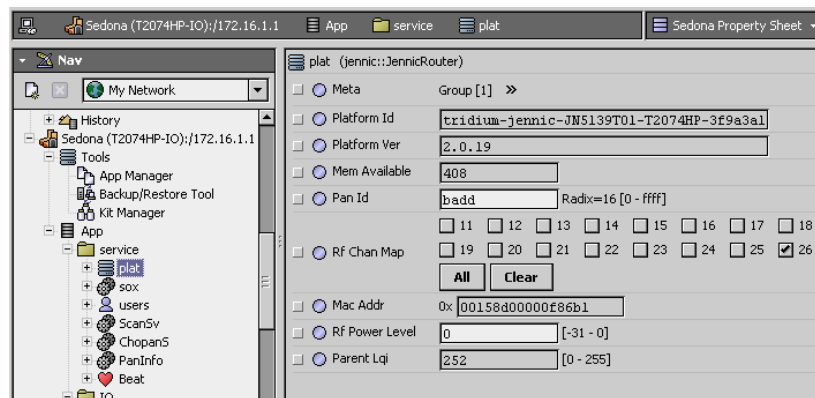
Note: *If the connection fails, it is likely because your Workbench host does not have the necessary manifest files for all the kits installed in the Sedona Framework device. In the error, examine any "Details" link to see if missing manifests are named. For related details, refer to Sedona Manifest Manager - Engineering Notes. Another possible issue might be a user mismatch between the app's users service and your station login. As a workaround, in the JACE station's SoxTunnel service (Config > Services > SoxTunnel), set the "Authenticate with User Service" property to false and Save.*

Figure 2-22 Successful Sox tunnel connection in Workbench



As shown above, the tunneled Sox connection also appears in the Nav tree root of that host (on parity with the station). You can also work from the Nav tree, expanding and right-clicking, etc. as needed.

Figure 2-23 Expanding tunneled Sox connection in Nav tree



[Figure 2-23](#) above shows the service folder expanded, with the **Sedona Property Sheet** of the "plat" component opened in the Workbench view. Click on folders in the App for wire sheet views.

Note: *The **App** icon changes to add a star whenever you have unsaved changes—meaning they have not been saved to the device's non-volatile Flash memory. Right-click the **App** and select **Actions > Save**. Otherwise, changes will remain in device RAM only, and be lost upon a reboot or power event.*

You can also use the Sox **Tools** when tunnel connected to the device. For related details, see the document *Sedona Framework Sox Tools Guide*.

CHAPTER 3

Niagara Sedona Framework Concepts

This section describes the NiagaraAX integration of Sedona Framework devices. These are the main subsections:

- “Understanding the different levels of data access” on page 3-1
- “About Sedona Framework network types” on page 3-2
- “About the SedonaNetwork component” on page 3-5
 - “SedonaNetwork properties” on page 3-6
 - “Sedona Device Manager view” on page 3-7
 - “Manifest Manager view of SedonaNetwork” on page 3-9
- “About the SedonaDevice component” on page 3-10
- “About the SedonaJen6lpNetwork component” on page 3-11
 - “About Jennic-based devices” on page 3-11
 - “About the Jennic RF network” on page 3-13
 - “SedonaJen6lpNetwork properties” on page 3-14
 - “SedonaJen6lpDevice actions” on page 3-25
 - “Sedona Jen6lp Device Manager view” on page 3-20
 - “Manifest Manager view of SedonaJen6lpNetwork” on page 3-21
 - “Pan Sheet view of SedonaJen6lpNetwork” on page 3-22
- “About the SedonaJen6lpDevice component” on page 3-23
 - “SedonaJen6lpDevice properties” on page 3-23
 - “SedonaJen6lpDevice actions” on page 3-25
- “About the Chopan Virtual gateway” on page 3-26
 - “Required configuration in the Jennic-based device” on page 3-26
 - “Walk-through using Chopan Virtual gateway” on page 3-27
 - “Additional Chopan usage topics” on page 3-32
- “About the Sedona Point Manager” on page 3-33
 - “Unique things about the Sedona Point Manager” on page 3-33
 - “Sedona Point Manager “Discovered” notes” on page 3-34
 - “Add dialogs in Sedona Point Manager” on page 3-35
 - “Sedona Point Manager “Database” notes” on page 3-37
- “About Sedona proxy points” on page 3-38
 - “Sedona and Niagara data types and null notes” on page 3-39
- “About Sedona action points” on page 3-41

Understanding the different levels of data access

Sedona Framework networks in a NiagaraAX station are unique in that (typically) Workbench provides at least two different levels of data access for an integrated Sedona Framework device:

- The (upper) Niagara component level, using NiagaraAX drivers with components that model the devices, and whatever data items were selected using Sedona proxy points and action points—standard NiagaraAX “station” data. The Niagara station communicates to Sedona Framework devices using the Sox protocol at the application layer, with lower layers either Ethernet/IP (or 802.11 WiFi) if a **SedonaNetwork** or Jennic 802.15.4 / 6LoWPAN (wireless) if a **SedonaJen6lpNetwork**. This document is mainly about this data access level.

Note: To be able to discover and add Sedona proxy points, the JACE or Supervisor (station host) requires the kit manifest file for each Sedona kit installed in any Sedona Framework device. Do this in a station connection to the host, using the “Manifest Manager” view on the Sedona Framework network.

- The (lower) native Sedona component level, using a “tunneled” Sox connection from Workbench through the running station. The station host (typically JACE) must be licensed to allow Sox tunneling. Also, the Workbench host must be licensed for provisioning. See [“Configure to allow Sox tunneling”](#) on page 2-13.

Once Workbench is Sox tunnel connected to a Sedona Framework device, Sedona provisioning of it may be possible, as well as making Sedona “app” changes to its components, including properties and links.

Note: *To be able to open a Sedona Framework device in a Sox connection, the provisioning Workbench PC (host) requires the kit manifest file for each Sedona Framework kit installed in the device. This applies whether “Sox tunneling” through the JACE station, or using a Jennic USB coordinator to open a Sox connection directly to a Jennic-based device.*

For related details, refer to the documents *Sedona Framework Manifest Manager - Engineering Notes* and *Sedona Sox Tunneling - Engineering Notes*.

- For SedonaJen6lpNetworks (only): An available “CHoPAN” (or simply Chopan) component access level, using a client-server architecture. Chopan provides Sedona proxy point polling advantages, providing that Jennic-based devices are configured and enabled for Chopan server operation. Chopan also allows a device’s Sedona Framework app to directly access data in other networked devices, or in the JACE station (providing they are Chopan servers) via *client-side* “Chopan points”. For *hibernating* Jennic-based devices, Chopan client configuration (Chopan points) must be used to retrieve all data, instead of using Sedona proxy points.

Note: *At the time of this document, Sedona Framework support for hibernating devices (typically battery powered devices) is not widely available. However, the SedonaJen6lpNetwork driver in the NiagaraAX station is “ready” for such device support if this changes.*

Client-side components in a Jennic-based device’s app are added via Niagara access to the JACE station, in the “Chopan Virtual” gateway under its corresponding SedonaJen6lpDevice component. See [“About the Chopan Virtual gateway”](#) on page 3-26 for related details. This document summarizes various Chopan-related components and views in the explanation of the SedonaJen6lpNetwork, but complete Chopan details are in the *Sedona Framework Chopan Usage* document.



Caution

In general, before engineering the upper (station) level with Niagara Sedona proxy points and action points, it is recommended to develop and complete the (native) Sedona Framework app running in each Sedona Framework device. Otherwise, data address mismatches between Sedona proxy points in the station and the source Sedona Framework components in a device are likely to occur.

About Sedona Framework network types

Starting in AX-3.6, there are two types of Sedona Framework networks possible in a NiagaraAX station: **SedonaNetwork** (for Ethernet/IP and/or WiFi-capable Sedona Framework devices) and **SedonaJen6lpNetwork** (for 802.15.4 wireless Jennic-based devices). Both network types use the standard NiagaraAX network architecture. See “About Network architecture” in the *Drivers Guide* for general information.

Note: *In previous AX-3.5 and AX-3.4 releases, the **SedonaNetwork** was sourced from a different module (nsedona), and could also integrate wireless Jennic-based devices, using a “Jennic6LowpanBridgeService” from the jennic module. This architecture changed in AX-3.6, with a separate **SedonaJen6lpNetwork** (jen6lp module) now used for Jennic-based device integration.*

- *SedonaNetworks created with AX-3.4 and AX-3.5 are NOT COMPATIBLE with Sedona Framework TXS 1.1 and AX-3.6!*
- *An offline conversion tool for making a station’s database .bog file compatible with the new Sedona architecture is available. Before upgrading any JACE with a SedonaNetwork to AX-3.6, you should use this tool to create a compatible station .bog file, so that you can install it during the upgrade commissioning process. For details, refer to the document NiagaraAX Station Migration Tool.*

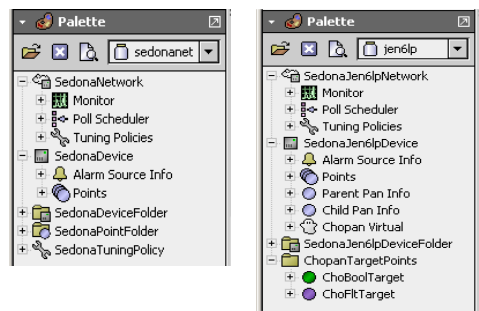
Sedona Framework networks: quick comparison

Sedona Framework network components and child components are found in the

- sedonanet palette — for **SedonaNetwork**
- jen6lp palette - for **SedonaJen6lpNetwork**

[Table 3-1](#) compares the palette for each module, as opened in Workbench.

Table 3-1 Palettes for modules sedonanet (left) and jen6lp (right)



Note: As with most other NiagaraAX drivers, you typically do not need to work from a module's palette. Instead, the various Sedona manager views simplify component creation, enforcing proper component hierarchy. A possible exception is use of the two “ChopanTargetPoints” in a SedonaJen6lpNetwork, which are “convenience” components that can be useful when making “target points” in the JACE to receive read-only values from a Jennic-based device, via CHoPAN.

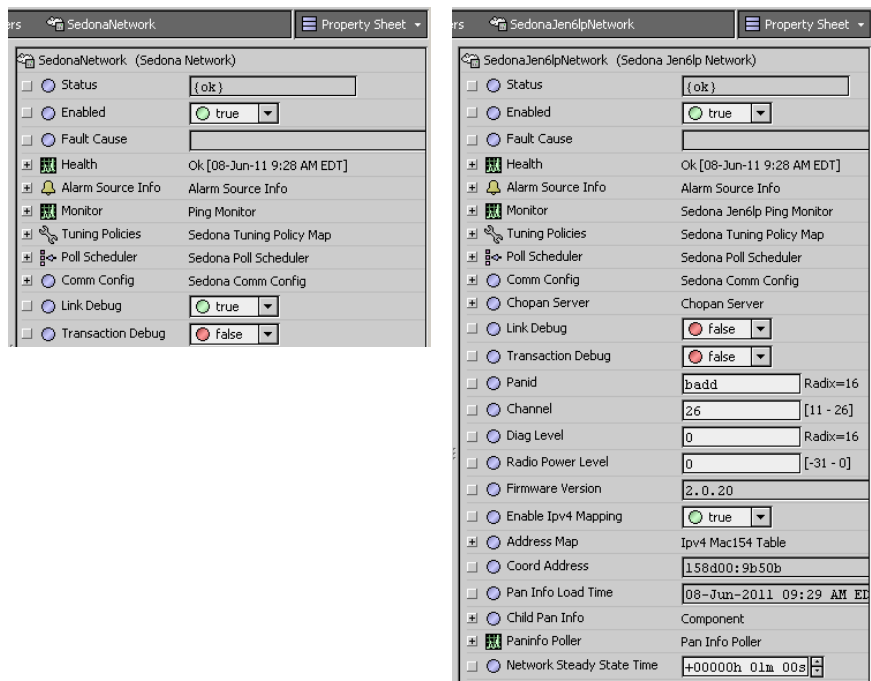
More differences and similarities between the two Sedona Framework network types are provided below:

- [Sedona Framework network component differences](#)
- [Sedona device component differences](#)
- [Similarities in Sedona Framework network types](#)

Sedona Framework network component differences

From the *property sheet* of either type of Sedona Framework network, you have access to all the major network-level properties and container slots. See [Table 3-2](#).

Table 3-2 Property sheet for SedonaNetwork (left) and SedonaJen6lpNetwork (right)



As shown in [Table 3-2](#), the SedonaJen6lpNetwork (right) has many more properties than the SedonaNetwork (left). Most of these are for the Jennic 802.15.4 configuration of the Sedona Jennic option card in the host JACE controller, for operation as the “Jennic coordinator” for the network.

The SedonaJen6lpNetwork also has “Pan Info” related properties, as do its child SedonaJen6lpDevice components, providing “diagnostic” data useful for troubleshooting wireless Jennic communications.

Currently, the SedonaJen6lpNetwork is also the only Sedona Framework network to use “CHoPAN”, reflected by the “ChopanServer” container slot in the network’s property sheet. Child SedonaJen6lpDevice components also have a related “Chopan virtual” components. CHoPAN is a light-weight “sessionless” protocol that allows data sharing between devices (and the JACE), without requiring the session-based Sox protocol.

Both Sedona Framework networks have a default “device manager” view for adding and managing child devices. However, only the Sedona Jen6lp Device Manager has a “learn” mode that allows online device discovery. The SedonaJen6lpNetwork also provides a “Pan Sheet” view for graphically representing the Jennic wireless network’s node (tree) hierarchy, including all current “Pan Info” data for the JenNet network.

For more details on properties, actions, and views of Sedona Framework network components, see:

- [“About the SedonaNetwork component”](#) on page 3-5
- [“About the SedonaJen6lpNetwork component”](#) on page 3-11

Sedona device component differences

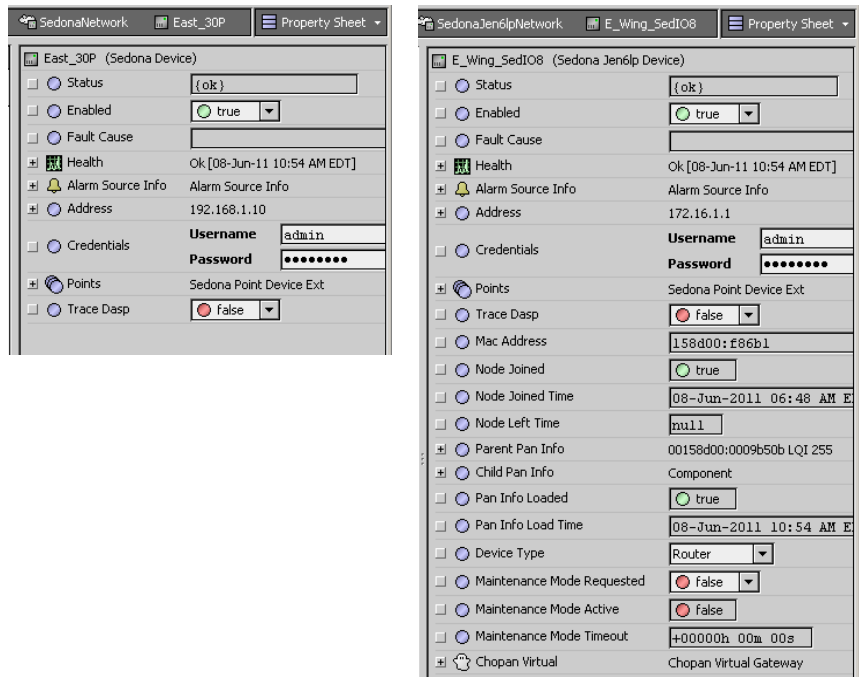
Device types differ between the two Sedona Framework networks—each network has only *one* valid device type:

- **SedonaDevice** — for SedonaNetwork
- **SedonaJen6lpDevice** — for SedonaJen6lpNetwork

You cannot put one device type into the opposite network type.

As shown in [Table 3-3](#), the SedonaJen6lpDevice (right) has more properties than the SedonaDevice (left).

Table 3-3 Property sheet for SedonaDevice (left) and SedonaJen6lpDevice (right)



Most additional properties of a SedonaJen6lpNetwork relate to its Jennic 802.15.4 operation, including “Pan Info” and “Maintenance Mode” data. The “Chopan Virtual Gateway” is for modeling the Chopan client interface in the Jennic-based device.

In addition to the standard “Ping” action on the SedonaDevice, a SedonaJen6lpDevice also has “Pan Info” actions and a “Request Maintenance Mode” action.

For more details on properties, actions, and views of Sedona device components, see:

- [“About the SedonaDevice component”](#) on page 3-10.
- [“About the SedonaJen6lpDevice component”](#) on page 3-23.

Similarities in Sedona Framework network types

Regardless of Sedona Framework network type, any Sedona device component has a single device extension: **Points**, for adding/managing Sedona proxy and action points. The **Sedona Point Manager** view on any device's Points extension has a "learn" mode that allows online point discovery.

Sedona proxy points (and action points) work in the same manner regardless of network type. Although Sedona proxy points in SedonaJen6lp device can use "Chopan" (as an alternate to Sox polling or event messaging) for updates to the JACE station, the efficiencies provided may mean more with wireless (Jennic) devices that it would with Ethernet/IP capable devices.

Sox tunneling to any Sedona Framework device, allowing "native" Sedona Framework app programming and/or Sox tools (provisioning), works in the same manner regardless of a device's Sedona Framework network type. However, if a hibernating¹ Jennic-based device in a SedonaJennic6lpNetwork, you must make a "maintenance mode" request (action) on that device to allow Sox access from Workbench.

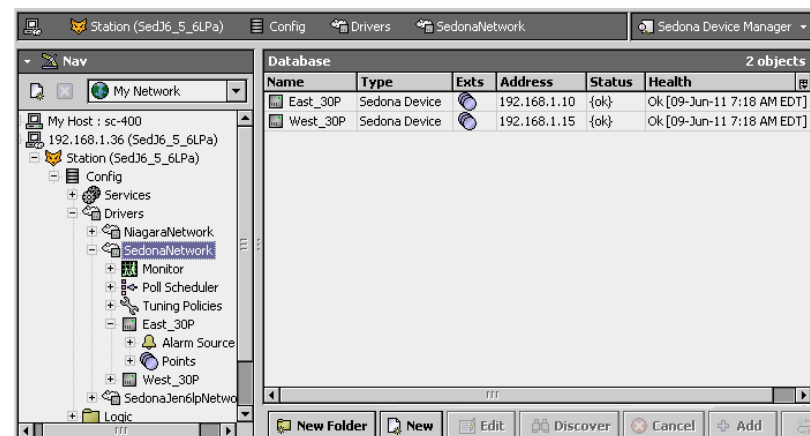
For more details on these topics common among Sedona Framework networks, see:

- ["About the Sedona Point Manager"](#) on page 3-33
- ["About Sedona proxy points"](#) on page 3-38
- ["About Sedona action points"](#) on page 3-41
- ["Configure to allow Sox tunneling"](#) on page 2-13

About the SedonaNetwork component

Starting in Sedona Framework TXS 1.1 and AX-3.6, the SedonaNetwork is the top-level component in a station for integration of Ethernet/IP and WiFi capable Sedona Framework devices (only). It is found in the sedonanet module. This network is licensed using the "sedonanet" feature, and may have device or point license limits.

Figure 3-1 SedonaNetwork (default Sedona Device Manager view)



The SedonaNetwork uses the typical NiagaraAX driver architecture, with standard components for device monitor, a poll scheduler, and tuning policies. In addition to the default **Sedona Device Manager** view shown in [Figure 3-1](#), there is also an available **Manifest Manager** view.

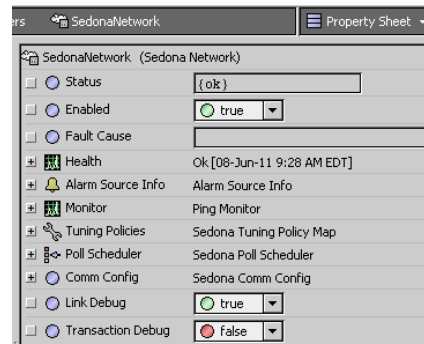
The following sections provide more SedonaNetwork details:

- [SedonaNetwork properties](#)
- ["Sedona Device Manager view"](#) on page 3-7
- ["Manifest Manager view of SedonaNetwork"](#) on page 3-9

1. Sedona Framework support for hibernating devices (typically battery powered devices) is not widely available in the initial Sedona Framework TXS release. However, the SedonaJen6lpNetwork driver in the NiagaraAX station is "ready" for such device support, including "maintenance mode" operation.

SedonaNetwork properties

Figure 3-2 SedonaNetwork property sheet



Among common properties, the **SedonaNetwork** has the usual collection of Niagara driver status, health, and (ping) monitor properties that work in the standard way. For more details, refer to the *NiagaraAX Driver Guide* section “Network status properties” and “About Monitor”.

Other common properties include the typical container slots for Tuning Policies and a Poll Scheduler. A single “Ping” action is available on the network.

SedonaNetwork tuning policy notes

Proxy points of a **SedonaDevice** (under a **SedonaNetwork**) can be assigned to a specific tuning policy, by name. By default, a single **SedonaTuningPolicy** named “Default Policy” is in the network’s Tuning Policies container. Typically, you should add additional tuning policies (duplicate, rename, modify properties) and assign to proxy points as needed. For general information about tuning policies and associated properties, see “About Tuning Policies” in the *NiagaraAX Driver Guide*.

Tuning policies of a **SedonaNetwork** have the following characteristics:

- The “Write On Start” and “Write On Enabled” properties have been modified to have a default value of *false*. This can help decrease network traffic, which may be critical with some devices.
- Point poll frequency is specified in the tuning policy (similar to how it is done for the BACnet driver).
- A “Comm Type” property specifies how Niagara attempts to update point values, where choices are:
 - Sox Event (default) — uses Sox to subscribe to points, where updates are sent via Sox event messages (similar to COV subscriptions in BACnet). See “[Sox Event considerations](#)”.
 - Sox Poll — uses Sox read messages to request point values.

Sox Event considerations Although Sox event subscriptions provide efficiencies over normal (Sox Poll) polling from Niagara, be aware that the default “Sox Event” may be inappropriate for some proxy points. Exercise caution if the property has a rapidly changing value, or is a property of a component that has *another property* with a rapidly changing value. Note the other property *does not have to be proxied*.

Why? A proxy point assigned to a Sox Event tuning policy results in subscription updates triggered by *any value change of any other “same type” property of that component* (type is “runtime” or “config”). This can be a problem if a rapidly-changing property value results in constant event subscription updates, adding unnecessary messaging and possible message fragmentation issues.

For example, consider a proxy point created for the “out” slot of a UI (universal input) component in the Sedona Framework app. Although its value may appear relatively stable, another “raw” property of that component is constantly churning a new A/D (analog to digital) value. Each change results in a new Sox event message update containing all “realtime” property values of that component (the “out” and “raw” values, plus any other realtime property values). This can lead to message bottlenecks on the network, as well as in the Sedona Framework device. In this case, it would be much better to assign a tuning policy using “Sox Poll” for the proxy point for the UI’s “out” slot.

Typically, this consideration applies more to “runtime” properties of a Sedona component, and not “config” properties. Ideal proxy point candidates for tuning policies using “Sox Event” comm type are slowly-changing boolean property types, where the parent Sedona component does not have another property with a rapidly changing value.

SedonaNetwork poll scheduler notes

The single Poll Scheduler of a **SedonaNetwork** has the usual collection of properties. For general information about the poll scheduler, see “About poll components” in the *NiagaraAX Driver Guide*.

SedonaNetwork Comm Config

Comm Config in the [SedonaNetwork properties](#) specifies communication parameters for Sedona. These parameters affect polls and writes sent to SedonaDevices.

- **Retry Count** — Number of times a failed message is retried before the transaction is abandoned, where the default value is 3.
- **Retry Time** — Elapsed time before giving up on a message send attempt.

SedonaNetwork debug properties

Among [SedonaNetwork properties](#) are two useful for debug purposes. If enabled, *and* you have station logSetup (in spy) set to “trace” for *SedonaNetworkName* and *SedonaNetworkName.sox*, this directs extra messaging to the station’s output (visible in platform Application Director):

- **Link Debug** — For additional trace-level data in station output, useful in tracing the Sox traffic on top of the DASP (Datagram Authenticated Session Protocol) traffic. For example:
1:46:29.529 PM|send0:127.0.0.01:r 72 02 00 07 01
1:46:31:485 PM|rcvd:127.0.0.1:R 52 02 00 07 01 06 42 10 00 00
- **Transaction Debug** — For additional trace-level data about Sox transaction management.

Sedona Device Manager view

Figure 3-3 Sedona Device Manager view



The **Sedona Device Manager** is the default view on a [SedonaNetwork](#), used to add and edit SedonaDevice components to represent remote Ethernet/IP-capable Sedona Framework devices. Online device discovery is *not supported*—you manually add devices using the **New** (and optionally **New Folder**) buttons.

This manager view operates in the standard way as other NiagaraAX drivers; see “About the Device Manager” in the NiagaraAX Drivers Guide for general information.

Note: The buttons **Discover**, **Cancel**, **Add**, and **Match** at the bottom of this view remain unavailable (dimmed), as this view has no “learn mode” with Discovered pane.

Other details on the Sedona Device Manager are in the following sections:

- [Adding SedonaDevice notes](#)
- [Sedona Device Manager data columns](#)

Adding SedonaDevice notes

To add SedonaDevice(s), click the **New** button to specify the number to add (type is always Sedona Device), then click **OK** for the **New** dialog, as shown in [Figure 3-4](#).

Note: Alternatively, you can first click **New Folder** to create a device folder (name it as needed) to organize devices. Then double-click that device folder for its own **Sedona Device Manager** view.

Figure 3-4 New dialog for adding a SedonaDevice

| Name | Type | Address | Enabled | Credentials |
|----------|---------------|--------------|---------|-----------------------|
| West_30P | Sedona Device | 192.168.1.15 | true | Username And Password |

☐ Name: West_30P
☐ Type: Sedona Device
☐ Address: Sedona Address
☐ Enabled: true
☐ Credentials: Username And Password

☐ Ip: 192.168.1.15
☐ Sox Port: 1876
☐ Chopan Port: 1811

Username: admin
Password:

OK Cancel

In the **New** dialog (or later, **Edit** dialog) for a SedonaDevice:

- IP address can be either the typical IPv4, or IPv6 (providing the JACE or Supervisor supports IPv6), for example: FE80:0000:0000:0000:0215:8D00:000E:41f7
- Sox port is typically left at the default 1876, unless it has been changed in the device's Sedona Framework app to another port—in which case, set it to match that port here.
- Credentials are typically left at the (default) admin user, unless user(s) have been modified in the device's Sedona Framework app. In that case, enter matching credentials here.

Note: For complete step-by-step details, see ["Add SedonaDevices"](#) on page 2-2.

Sedona Device Manager data columns

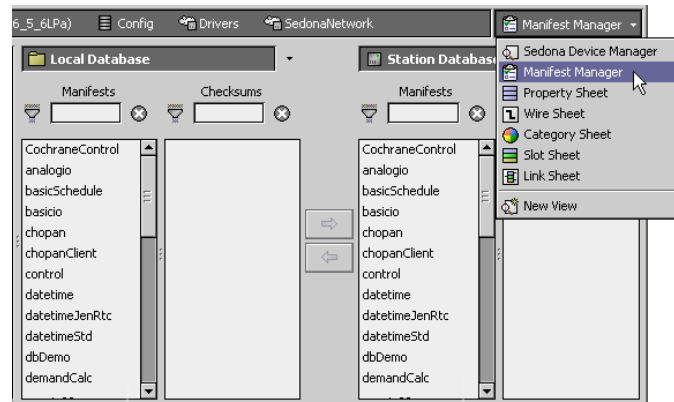
By default, table columns in the Database pane of the [Sedona Device Manager](#) include the following:

- **Name**
Niagara name for the SedonaDevice.
- **Type**
Always "Sedona Device" (cannot modify).
- **Exts**
 Points extension for the device — double-click for its **Sedona Point Manager** view.
- **Address**
IP address of the Sedona Framework device.
- **Status**
Device component status, typically "ok" but possibly "fault" (license issue) or "down".
- **Health**
"Ok" or "Fail" with timestamp of last device monitor ping. Fail typically coincides with a down status.
- **Fault Cause**
Typically blank, unless health is Fail, where any available "Fault Cause" string appears.

Note: Not included by default, but available in the *Table Options (menu)*, are items "Enabled" and "Credentials".

Manifest Manager view of SedonaNetwork

Figure 3-5 Manifest Manager is an available view on the SedonaNetwork



As shown in [Figure 3-5](#), the **Manifest Manager** is an available view on a [SedonaNetwork](#)—note it is the *same view* that is available on a [SedonaJen61pNetwork](#).

You typically use this view to copy Sedona kit manifest files (“manifests”) from your Workbench host (Local Database) to the JACE hosting the station (Station Database). In this case you copy manifests *from the left side to the right side*. The station running any Sedona Framework network needs direct access to all kit manifest files for any networked device, to support online Sedona proxy point discovery and proxy point updates. For a related quick start procedure in this document, see [“Run the Manifest Manager to ensure kit manifests are loaded”](#) on page 2-8.

However, this view also works “bi-directionally”. That is, you can use it to copy manifests *from the JACE* (or Supervisor running the SedonaNetwork) *to your Workbench computer*, copying from right to left. This may be useful if you access the station using a different Workbench host than was used to engineer the station. Note that Workbench requires direct access to the manifests for all kits in a Sedona Framework device when making a Sox connection to it (typically, a Sox tunneling connection)—otherwise, the connection will fail with an exception (exception error “details” usually list the missing manifest file or files).

Note: *Niagara Workbench with Sedona Framework TXS also provides a similar “upper level” **Sedona Manifest Manager** view, launched from the menu bar: **Tools > Sedona Manifest Manager**. In that view, the left pane is “Web Database” (source, read-only), while the right (target) pane is your Workbench computer. Providing that you have Internet connectivity, this gives access to manifest files available at sedonadev.org.*

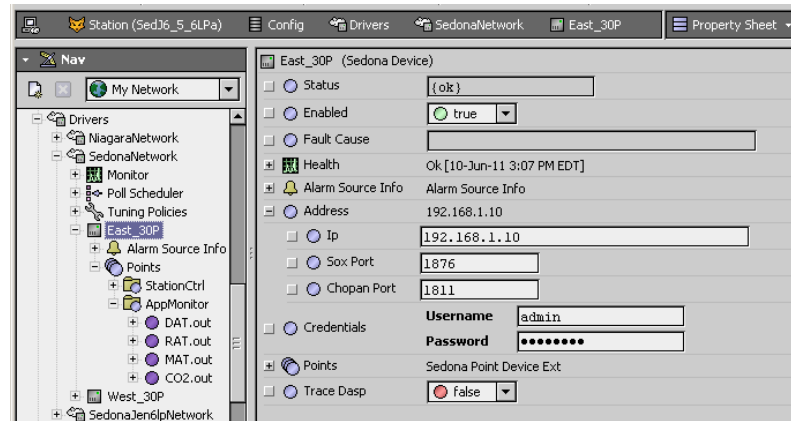
Find complete details about using the Manifest Manager in the *Sedona Framework Manifest Manager - Engineering Notes* document.

About the SedonaDevice component

The SedonaDevice is the device-level component representing an Ethernet/IP-capable (or WiFi capable) Sedona Framework device, and is a direct child of a [SedonaNetwork](#). It contains properties and slots typical to most driver's device components—see “Common device components” in the *Drivers Guide* for general information on device component status properties, health, and alarm source info.

[Figure 3-6](#) shows the property sheet view of a SedonaDevice—its default view.

Figure 3-6 SedonaDevice property sheet



Properties unique to the SedonaDevice include:

- **Address**
 - **Ip** — IP address of the Sedona Framework device, reachable by the JACE (NiagaraAX host). Can be either typical IPv4 address, for example 192.168.1.4, or (if supported by the NiagaraAX host) IPv6, for example: FE80:0000:0000:0000:0215:8D00:000E:41f7
 - **Sox Port** — The port the device's Sox service is listening on. Default port is 1876.
 - **Chopan Port** — The port the device's CHoPAN service is listening on. (Currently, CHoPAN is *not applicable* to SedonaDevices under the SedonaNetwork).
- **Credentials**
The credentials used to authenticate to the device in a Sox connection.
- **Points**
The single device extension, with a default **Sedona Point Manager** view. For related details, see “[About the Sedona Point Manager](#)” on page 3-33.
- **Trace Dasp**
If enabled, allows for device-specific tracing of the Sox traffic over the DASP (Datagram Authenticated Session Protocol) traffic.

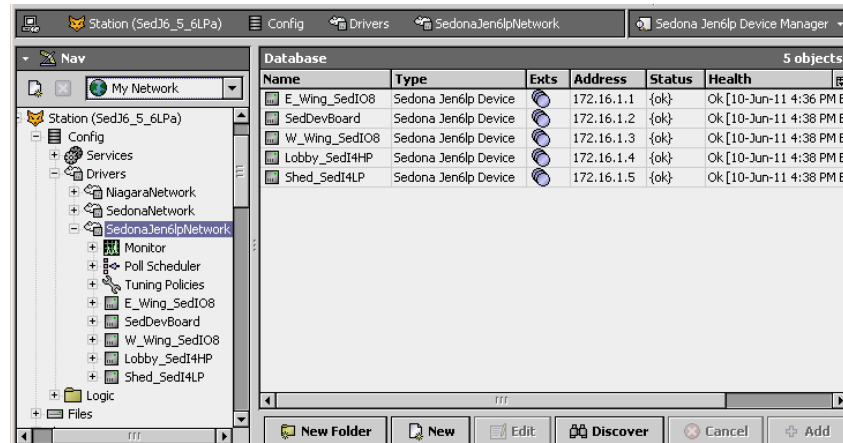
A single “Ping” action is available on the SedonaDevice. No special views are available on the SedonaDevice, apart from the standard ones (Wire Sheet, Category Sheet, Slot Sheet, Link Sheet).

About the SedonaJen6lpNetwork component

Starting in Sedona Framework TXS 1.1 and AX-3.6, the SedonaJen6lpNetwork is the top-level component in a JACE-2, -6, -7 or JACE-x02 XPR station for integration of wireless Jennic-based devices (only), and is found in the `jen6lp` module. This network is licensed using the “jen6lp” feature, and may have device or point license limits.

Note: *The JACE controller requires an installed Sedona Jennic option card to use this network, acting as the Jennic “coordinator” node to the wireless network of devices. Only one Sedona Jennic option card/ SedonaJen6lpNetwork is supported per JACE.*

Figure 3-7 Sedona Jen6lpNetwork (default Sedona Jen6lp Device Manager view)



The SedonaJen6lpNetwork uses the typical NiagaraAX driver architecture, with typical components for device monitor, a poll scheduler, and tuning policies. This network component extends the “base” SedonaNetwork with additional slots and views that support the “JenNet” protocol on the RF wireless 802.15.4 WPAN, as well as the 6LoWPAN/IP layer above that.

In addition to the default **Sedona Jen6lp Device Manager** view shown in [Figure 3-7](#), there is also an available **Manifest Manager** view and **Pan Sheet** view on the network. The following main sections provide more SedonaJen6lpNetwork details:

- “[About Jennic-based devices](#)” on page 3-11 (background)
- “[About the Jennic RF network](#)” on page 3-13 (background)
- “[SedonaJen6lpNetwork properties](#)” on page 3-14
- “[SedonaJen6lpNetwork actions](#)” on page 3-19
- “[Sedona Jen6lp Device Manager view](#)” on page 3-20
- “[Manifest Manager view of SedonaJen6lpNetwork](#)” on page 3-21
- “[Pan Sheet view of SedonaJen6lpNetwork](#)” on page 3-22

About Jennic-based devices

The term “Jennic-based” device is used in NiagaraAX and Sedona Framework documentation to mean a device based on a Jennic micro-controller, with built-in 802.15.4 wireless connectivity, 6LoWPAN stack support, and the Sedona Framework Virtual Machine (VM). Typical devices are low density I/O modules, thermostats, or other application-specific devices, often installed in places where running communications wiring (back to the JACE) is deemed cost prohibitive.

From a Niagara integration viewpoint, there are two important categories of Jennic-based devices:

- Continuously powered devices — Devices that are always powered, typically from some AC source. Unless noted otherwise, Jennic-based devices are assumed to be in this category.
- Hibernating devices — Devices typically powered by a self-contained battery or batteries. These devices introduce some integration complexity. See “[Hibernating devices](#)”.

Additionally, it can be helpful to understand details on the underlying Jennic RF network. See “[About the Jennic RF network](#)” on page 3-13.

Hibernating devices

Note: *At the time of this document, Sedona Framework support for hibernating devices is not widely available. Therefore this section does not apply to most installations. Other sections of this document that mention hibernating devices also note this. However, the SedonaJen6lpNetwork driver in the NiagaraAX station is “ready” for such devices, in case this changes.*

Some Jennic-based devices may be battery-powered, where the only power source is a self-contained battery (or batteries). To conserve battery life, such a device “hibernates” (sleeps) most of the time. At some periodic, configurable interval, the device “wakes up” and executes its routines, which include sending and receiving updates (as needed) on the wireless network. The device then immediately returns to hibernation, consuming almost no power.

While hibernating, the device is unreachable on the Jennic network. Here, the term “wireless” applies to both station communications and power wiring—as there is no power wiring to an external power source. Low power operation is *essential* to minimize the eventual need to replace these batteries.

To support hibernating device integration in Niagara, a special “sessionless” CHoPAN protocol (Compressed HTTP over Personal Area Network) is used for communications between the device and the JACE acting as the coordinator node. Associated are “Chopan client” components, configured and used in the device’s Sedona Framework app. In the station, each SedonaJen6lpDevice component includes a special “Chopan Virtual” gateway for building the necessary “Chopan points”.

More Chopan details are in other sections of this document, with complete details contained in the document *Sedona Framework Chopan Usage*.

Chopan is used in a Jennic-based hibernating device in solving these challenges:

- [Point updates \(read or write from/to station\)](#)
- [Allowing a Sox connection \(from Workbench\)](#)

Point updates (read or write from/to station) In normal operation, a hibernating device is in hibernation most of the time, and thus a poor candidate for standard Sedona proxy points. *Sedona proxy points should not be used for a hibernating device.* Even if the device is configured as a Chopan server (which is not recommended), and proxy points are assigned to a tuning policy using Chopan “comm type”, most poll requests will go unanswered because the device is hibernating. This results in unanswered poll requests, retries, and polling timeouts. Thus, it can adversely affect point polling of other continuously powered devices.

Instead of proxy points, you use the “Chopan Virtual” gateway of a SedonaJen6lpDevice to model “Chopan points” in the device’s Sedona Framework app. Chopan points can be either Boolean or Numeric types, and read-only or writable. Additional Sedona Framework app programming is then required to “link in” Chopan points to other components in the app. In the Niagara station, standard control point “targets” can be made to “receive” writable points from the Sedona Framework app. Conversely, writable Niagara points can write to read-only Chopan points in the hibernating device.

For related details, see [“About the Chopan Virtual gateway”](#) on page 3-26.

Allowing a Sox connection (from Workbench) In normal operation, the periodic “awake time” of a hibernating device is often a fraction of a second—again, to conserve precious battery power. However, sometimes a device needs to remain awake, for example when some change is needed in its Sedona Framework app, or if Sedona Framework provisioning (kit changes, backups, etc.) is required.

A Sox connection from Workbench is needed to perform such Sedona Framework operations. When you use Workbench to Sox connect to a hibernating device, the device automatically remains awake for the duration of that session. When you Sox disconnect from the device, it returns back to hibernation.

However, the normal “awake time” of a hibernating device is too short to set up a Sox connection. The device needs a way to “know” that a Sox connection is needed, so it can suspend hibernation for some period, to allow a Sox connection. To provide this, the SedonaJen6lpDevice component in the JACE station (that represents any Jennic-based device) has an available action “Request Maintenance Mode”.

For related details, see [“About maintenance mode”](#) on page 3-26.

About the Jennic RF network

This section summarizes the underlying architecture used for a network of devices represented in Niagara by a SedonaJen6lpNetwork. Some oversimplifications exist. For more complete details about Jennic 802.15.4 wireless communications, refer to Jennic documentation. At the time of this document, one URL for accessing this information is: www.jennic.com/support/user_guides

- “Jennic-based device protocol stack” on page 3-13
- “Jennic radio operation notes” on page 3-13

Jennic-based device protocol stack

Jennic-based devices use a protocol stack, that from “bottom up” use the following:

- IEEE 802.15.4 at the bottom physical layer and MAC (media access control) layer above—using wireless RF in the 2.4Ghz frequency range. This protocol defines the lower layers of a WPAN (wireless personal area network), including 16 “channels” and the definition of one node being a “full function” type with PAN “coordinator” ability, that is, in charge of the whole network.
- “JenNet”, the Jennic protocol that manages network formation and message routing, is the next layer up. A JenNet network has nodes that all use the same “PAN ID” (a two byte hexadecimal number). JenNet uses a “self healing tree” network structure, where all messaging goes up and down the tree. Nodes can have only one parent, but may have one or more children. Peer nodes do not communicate directly with each other, but rather through their parent.
Every 802.15.4 network has a single coordinator node plus one or more routers and/or end devices. In a Jennic-based network, the JACE is always the coordinator, at the top of tree. Routers are continuously powered Jennic-based devices capable of relaying messages between nodes (typically in addition to performing other application functions). End devices cannot relay messages, and may be Jennic-based hibernating devices.
Nodes maintain “PAN info” about any children, and routers also maintain this information about their parent. End devices do not store any “PAN info”. In the Niagara station, this “PAN info” is available graphically as a view on the SedonaJen6lpNetwork, and numerically in properties of some components. For details in this document, see “[Pan Sheet view of SedonaJen6lpNetwork](#)” on page 3-22. For more details, see the engineering notes document *Jennic Network Visualization (Pan Sheet)*.
- Above the JenNet protocol layer is the 6LoWPAN layer, which the Jennic-based device uses to establish an IPv6 address. Through the operation of the JACE coordinator and Jennic-based driver, each child node under the coordinator also automatically has a mapped IPv4 “private network” address. The IPv4 address is used to support Workbench “tunneled Sox” connections to devices.
- Sedona Framework is the application layer, which includes protocols DASP (Datagram Authentication Session Protocol) and the Sox protocol, along with other parts of the Sedona Framework.

Jennic radio operation notes

Because Jennic-based devices operate in the 2.4Ghz band spectrum, common to other wireless devices like 802.11 WiFi, Bluetooth, cordless phones, and baby monitors, a few guidelines may help when configuring a Jennic-based network, to minimize interference. Additional considerations can also help optimize throughput.

Location of devices Jennic-based devices, including the JACE with Sedona Jennic option card, should not be located near noise-generating equipment such as microwave ovens. Other devices based on personal area networks (PAN) may also cause interference, as well as devices previously mentioned.

Use of an antenna extension cable is often required, for a number of possible reasons—often more than one. Examples include to relocate the device’s antenna outside of a metal enclosure that houses it, to distance it from the antenna of another device, and/or to optimize reception and transmission.

Acceptable distances between the (JACE) coordinator and nodes will vary with a number of factors, including building structure makeup, sources of interference, and so on. Use of the Pan Sheet view on the SedonaJen6lpNetwork can help diagnose the JenNet tree structure and the link qualities between nodes.

SedonaJen6lpNetwork (Sedona Jennic option card) configuration Several network properties can affect radio operation; one example is Channel selection. For more details, see “[SedonaJen6lpNetwork coordinator properties](#)” on page 3-16.

SedonaJen6lpNetwork properties

Among common properties, the [SedonaJen6lpNetwork](#) has the usual collection of Niagara driver status, health, and (ping) monitor properties. For general details, refer to the *NiagaraAX Driver Guide* section “Network status properties” and “About Monitor”.

Figure 3-8 SedonaJen6lpNetwork property sheet

| SedonaJen6lpNetwork (Sedona Jen6lp Network) | |
|---|----------------------------|
| Status | {ok} |
| Enabled | true |
| Fault Cause | |
| Health | Ok [08-Jun-11 9:28 AM EDT] |
| Alarm Source Info | Alarm Source Info |
| Monitor | Sedona Jen6lp Ping Monitor |
| Tuning Policies | Sedona Tuning Policy Map |
| Poll Scheduler | Sedona Poll Scheduler |
| Comm Config | Sedona Comm Config |
| Chopan Server | Chopan Server |
| Link Debug | false |
| Transaction Debug | false |
| Panid | badd Radix=16 |
| Channel | 26 [11 - 26] |
| Diag Level | 0 Radix=16 |
| Radio Power Level | 0 [-31 - 0] |
| Firmware Version | 2.0.20 |
| Enable Ipv4 Mapping | true |
| Address Map | Ipv4 Mac154 Table |
| Coord Address | 158d00:9b50b |
| Pan Info Load Time | 08-Jun-2011 09:29 AM ET |
| Child Pan Info | Component |
| Paninfo Poller | Pan Info Poller |
| Network Steady State Time | +00000h 01m 00s |

Note: The network’s Monitor mechanism, as well as the related “Ping” action on each child *SedonaJen6lpDevice*, is unique in that device communications are not used on a ping. Instead, the check is with the “Joined” status of the device in the JACE coordinator’s address map for the JenNet network. See these automatically-added entries by expanding the “Address Map” container slot in the property sheet of the *SedonaJen6lpNetwork*. For details, see [“SedonaJen6lpNetwork coordinator properties”](#) on page 3-16.

Other common properties include the typical container slots for Tuning Policies and a Poll Scheduler. A single “Ping” action is available on the network.

These sections provide more details on *SedonaJen6lpNetwork* properties:

- [“SedonaJen6lpNetwork tuning policy notes”](#) on page 3-14
- [“SedonaJen6lpNetwork poll scheduler notes”](#) on page 3-16
- [“SedonaJen6lpNetwork Comm Config”](#) on page 3-16
- [“SedonaJen6lpNetwork Chopan Server”](#) on page 3-16
- [“SedonaJen6lpNetwork debug properties”](#) on page 3-16
- [“SedonaJen6lpNetwork coordinator properties”](#) on page 3-16
- [“SedonaJen6lpNetwork Pan Info properties”](#) on page 3-18
- [“SedonaJen6lpNetwork Network Steady State Time property”](#) on page 3-19

SedonaJen6lpNetwork tuning policy notes

Proxy points of a *SedonaJen6lpDevice* (under a *SedonaJen6lpNetwork*) are assigned to a specific tuning policy, by name. By default, a single *SedonaTuningPolicy* named “Default Policy” is in the network’s Tuning Policies container, with a “Comm Type” of “Chopan”. Sometimes, this is not appropriate.

Typically, you should *add additional tuning policies* (duplicate, rename, modify properties) and assign to proxy points as needed. For general information about tuning policies and associated properties, see “About Tuning Policies” in the *NiagaraAX Driver Guide*.

Tuning policies of *SedonaJen6lpNetwork* have the following characteristics:

- The “Write On Start” and “Write On Enabled” properties have been modified to have a default value of *false*. This can help decrease network traffic, which may be critical with some devices.
- Point poll frequency is specified in the tuning policy (similar to how it is done for the BACnet driver).
- A “Comm Type” property specifies how Niagara attempts to update point values, where choices are:
 - Chopan — (default) uses CHoPAN “GET” requests to retrieve point values. This provides effi-

ciencies over other (Sox) comm types, by “batching” multiple reads into a single request.

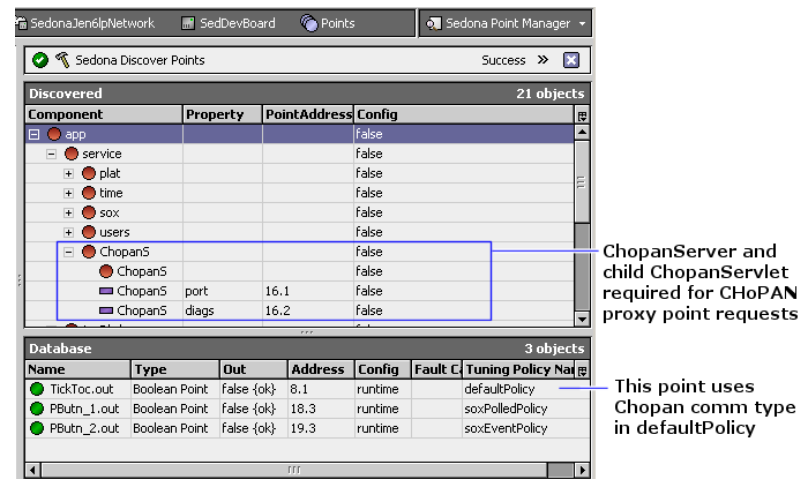
Note: Proxy points using a tuning policy with the Chopan comm type will remain in fault unless the associated Jennic-based device has the chopan kit installed, and its Sedona Framework app is configured to be a Chopan server. For related details, see “Chopan comm type considerations”.

- Sox Event — uses Sox to subscribe to points, where updates are sent via Sox event messages (similar to COV subscriptions in BACnet). See “Sox Event comm type considerations”.
- Sox Poll — uses individual Sox read messages to request point values.

Chopan comm type considerations Chopan is the most efficient “Comm Type” for a tuning policy, providing a device is configured as a Chopan server. Otherwise, you will need to assign a tuning policy using a comm type of either “Sox Poll” or “Sox Event” to a device’s proxy points. Of those two choices, “Sox Poll” is best for rapidly changing values—see “Sox Event comm type considerations” for details.

In the initial point discovery of a Jennic-based device, you can determine if the device supports CHoPAN message requests, by looking in the Sedona Point Manager’s “Discovered” pane (Figure 3-9).

Figure 3-9 Look for ChopanS(ervice) and child ChopanS(ervlet) under App’s services node



Expand the **app** node then **service** node, and look for “ChopanS” with a child “ChopanS”, as shown in Figure 3-9 above. These represent the required ChopanService and ChopanServlet, respectively. If these components are not found, likely the device is not configured to support tuning policies using Chopan.

In this case, proxy points using a tuning policy with “Chopan” as comm type will remain in fault.

Sox Event comm type considerations Although Chopan is typically the most efficient “Comm Type” for a tuning policy, it may be that a target Jennic-based device may not be configured to support it. If not, you must assign proxy points to a tuning policy using another comm types (Sox Event or Sox Poll).

Sox event subscriptions provide efficiencies over normal (Sox Poll) polling from Niagara. However, be aware that the “Sox Event” choice may be inappropriate for some proxy points. Exercise caution if the property has a rapidly changing value, or is a property of a component that has *another property* with a rapidly changing value. Note that the other property *does not have to be proxied*.

Why? A proxy point assigned to a Sox Event tuning policy results in subscription updates triggered by *any value change of any other “same type” property of that component* (type is “runtime” or “config”). This can be a problem if a rapidly-changing property value results in constant event subscription updates, adding unnecessary messaging and possible message fragmentation issues.

For example, consider a proxy point created for the “out” slot of a UI (universal input) component in the Sedona Framework app. Although its value may appear relatively stable, another “raw” property of that component is constantly churning a new A/D (analog to digital) value. Each change results in a new Sox event message update containing all “realtime” property values of that component (the “out” and “raw” values, plus any other realtime property values). This can lead to message bottlenecks on the network, as well as in the device. In this case, it would be much better to assign a tuning policy using “Sox Poll” for the proxy point for the UI’s “out” slot. Even more efficient would be “Chopan” (if supported by the device).

Typically, this Sox Event consideration applies more to “runtime” properties of a Sedona component, and not “config” properties. Ideal proxy point candidates for tuning policies using “Sox Event” comm type are boolean property types, where the parent Sedona component does not have another “like type” property with a rapidly changing value.

SedonaJen6lpNetwork poll scheduler notes

The single Poll Scheduler of a SedonaJen6lpNetwork has the usual collection of properties. For general information about the poll scheduler, see “About poll components” in the *NiagaraAX Driver Guide*.

SedonaJen6lpNetwork Comm Config

Comm Config in the [SedonaJen6lpNetwork properties](#) specifies communication parameters for Sedona. These parameters affect polls and writes sent to Jennic-based devices.

- **Retry Count**
Number of times a failed message is retried before the transaction is abandoned, where the default value is 3.
- **Retry Time**
Elapsed time before giving up on a message send attempt.

SedonaJen6lpNetwork Chopan Server

Chopan Server in the [SedonaJen6lpNetwork properties](#) contains properties to configure the CHoPAN server mechanism for this network in the JACE's station. Networked Jennic-based devices can access station data via “Chopan points” in their apps, as *client requests* to this server. Chopan server operation is required if the network includes any hibernating¹ Jennic-based devices.

Note: *Chopan server operation requires the JACE's jen6lp license feature to have attribute export="true".*

- **Port**
UDP port that the JACE station monitors for CHoPAN client requests from networked devices. The default port is 1810.
- **Enabled**
Enables or disables (the default) the network's Chopan server.
- **Debug On**
If enabled, *and* you have station logSetup (in spy) set to “trace” for `SedonaJen6lpNetwork.chopanSvr`, extra messaging is directed to the station's output (visible in platform Application Director).
- **Status**
Status of station's Chopan server, typically “ok” if enabled.
Note: *Status is fault if the JACE's license feature jen6lp does not have attribute export="true".*
- **Fault Cause**
If the Chopan server has a fault status, this text string explains why.

SedonaJen6lpNetwork debug properties

Two [SedonaJen6lpNetwork properties](#) are useful if troubleshooting. If enabled, *and* you have station logSetup (in spy) set to “trace” for `SedonaJen6lpNetworkName` and `SedonaJen6lpNetworkName.sox`, this directs extra messaging to the station's output (visible in platform Application Director):

- **Link Debug**
For additional trace-level data about Sox traffic on top of the DASP (Datagram Authenticated Session Protocol) traffic. For example:
1:46:29.529 PM|send0:127.0.0.01:r 72 02 00 07 01
1:46:31:485 PM|rcvd:127.0.0.1:R 52 02 00 07 01 06 42 10 00 00
- **Transaction Debug**
For additional trace-level data about Sox transaction management.

SedonaJen6lpNetwork coordinator properties

Among [SedonaJen6lpNetwork properties](#) are several that apply to the Jennic coordinator operation of the Sedona Jennic option card in the JACE controller:

- **Panid**
This PAN ID is a two-byte identifier from 0000 to ffff (in hexadecimal) that must be set to match the one in the Sedona Framework app of each Jennic-based device. The default Panid value is “defa”. For jobs that have multiple JACEs equipped with a Sedona Jennic option card, it is important to use a different PAN ID for each one (SedonaJen6lpNetwork).
- **Channel**
The Sedona Jennic option card for a JACE operates in the 2.4Ghz range, on one of 16 selectable channels. Channels are from 11 to 26. Many channels overlap channels used in 802.11 WiFi. To avoid interference from WiFi (in U.S. installations), recommended channels are 15, 20, 25, or 26.

1. Currently, Sedona Framework support for hibernating devices is not widely available. Therefore, use of the JACE Chopan server is not typically required.

Note: Jennic-based devices to be integrated typically have a “Channel Map” property in their Sedona Framework app (e.g. `service.plat.rfChanMap`) that specifies which of the 16 possible channels it searches (with “beacon” message) for the network coordinator. A best practice is to limit the value of this property to only the channel used by the JACE option card (to which the device will be networked).

In jobs with multiple JACEs equipped with a Sedona Jennic option card, it is recommended to use unique channel assignments, if possible, so as not to share bandwidth.

- **Diag Level**

The level of Jennic/Sedona Framework-related debug messaging to include in the station’s standard output (in the platform Application Director view). The value of 0 (default) provides the smallest level. This is read as a bit field, where each bit in the field turns on a different set of debug.

Note: The station’s log level must have `sedona.6lp` and `sedona.plat.jen6lp` set to trace level in order for this property to have any effect. To minimize resource overhead, return log levels back to message level after debugging.

- **Radio Power Level**

This controls attenuation of the RF transmit power level of the coordinator, where the default (0) is highest power. Range is in db, from minimum (-31) to maximum (0).

In most cases, radio power level is best left at default. If desired, you can set to lower power levels to “simulate” devices that are farther apart or have poor RF connectivity.

- **Firmware Version**

The firmware version level of the Sedona Jennic option card installed in the JACE.

Note: Option card firmware is automatically installed the first time a station starts up with a new or upgraded `platJen6lp` module. Typically this adds about 30 seconds to the station startup process.



Caution

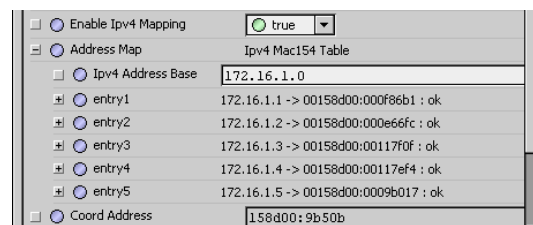
Do not remove power to the JACE during this initial station startup process, for example following an upgrade of software that includes the `platJen6lp` module. Otherwise, it could be possible for the option card to lose important data.

- **Enable Ipv4 Mapping**

Enables mapping of IPv6 addresses of devices to IPv4 decimal addresses (default is true).

- **Address Map**

Figure 3-10 Example Address Map entries after enabling network



Container to specify the IPv4 address network “base” as well as hold entries for discovered nodes:

- **Ipv4 Address Base**

Private IPv4 subnet for the JACE coordinator to map discovered wireless Jennic-based devices. This subnet must fall within the Class A, B, or C address range, as follows:

- 10.0.0.0 to 10.255.255.0 (Class A)
- 172.16.0.0 to 172.31.255.0 (Class B)
- 192.168.0.0 to 192.168.255.0 (Class C)

Note: This must be a different IP subnet than the one used by the JACE. Otherwise, the JACE TCP/IP stack will be unable to route packets appropriately. Also, this must be an unused subnet on the domain. For example, if the domain uses the 192.168.1.0 subnet, the default Ipv4 Address Base should not be used. Use a subnet not utilized, say 10.10.8.0, or 172.16.1.0.

- **entry1-n**

Automatically added entries for Jennic-based devices discovered by the JACE coordinator. As new wireless devices are discovered, they are assigned the next available IP address. The first address ($n.n.n.0$) is reserved for the coordinator.

Each of these dynamic entries has the following read-only properties:

- **Ipv4** — Mapped IPv4 address for the device, which is used as its Address when discovered and added as a `SedonaJen6lpDevice` in the network.
- **Mac154** — The unique 802.15.4 64-bit (8 byte) MAC address of the device, in hexadecimal notation, for example: 00158d00:000f86b1

- Joined — Current “JenNet” status with the JACE coordinator, where “true” reflects “ok” device status, else “false” means no communications (device seen as “down”).

***Note:** These dynamic entries persist until cleared. Note a “Clear Table” action is available on the Address Map container, which clears all dynamic entries—potentially useful if “starting over” with a new address base.*

- **Coord Address**

The unique 802.15.4 MAC address of the Sedona Jennic option card (JACE coordinator), in hexadecimal notation without leading zeroes, for example: 158d00:9b50b

SedonaJen6lpNetwork Pan Info properties

Among [SedonaJen6lpNetwork properties](#) are several applicable to JenNet “PAN info” for networked wireless devices, including link quality and statistical data. This data, along with data from child devices operating as router nodes, is used in the network’s graphical **Pan Sheet** view.

***Note:** PAN info requires each router-capable device to have the paninfo kit. If PAN info is to be read by CHoPAN, the device also requires the chopan and paninfoChopan kits. Some minor app configuration is required. Devices that are not router-capable do not require any configuration. Please refer to the “PAN Info Quick Steps” section in the Jennic Network Visualization (Pan Sheet) - Engineering Notes document.*

SedonaJen6lpNetwork Pan Info properties include:

- **Pan Info Load Time**

Timestamp of last successful load of PAN info.

- **Child Pan Info**

Container component for dynamically-added SedonaJen6lpNeighborEntry (MAC802.15.4MacAddress) child components, described below.

- **MAC802.15.4MacAddress LQI *n*** (for example, MAC00158d00_00117f0f LQI 174)
Contains read-only data about each direct child node under the JACE coordinator, including:
 - Is Sleeping End Device — Whether device is a battery powered (hibernating) end device type (`true`) or a continuously powered device (`false`).
 - Link Quality — Value between 0-255, representing strength of last packet received from this node. Values over 60 represent a good link. Also known as “LQI” (Link Quality Index).
 - Packets Lost — Number of packets sent to device for which an ack(nowledgement) was not received. Note an ack is not expected on all packets.
 - Packets Sent — Number of packets sent to device from coordinator.
 - Packets Received — Number of packets coordinator has received from device.
 - Mac154 — The unique 802.15.4 MAC address of the device, in hexadecimal notation without leading zeroes, for example: 158d00:9b017
 - Stale — Typically false, else true if this node entry is stale, meaning the device represented is no longer a child. For example, if a device resets and rejoins the JenNet network, it may have a new parent (say a router node). In this case, this Stale values shows true.

- **Pan Info Poller**

***Note:** PAN info polling may be useful for debugging, but increases network traffic.*

Contains configuration and read-only status properties for the “PAN info polling” mechanism that retrieves JenNet PAN info data from child Jennic-based devices on the SedonaJen6lpNetwork.

- **Enabled**

Enables PAN info polling if `true`, else if `false` (the default) disables polling.

- **Use Chopan**

Whether CHoPAN is used for PAN info polling (the default, `true`) or if `false`, Sox messaging is used for PAN info polling.

- **Desired Poll Period**

Desired PAN info poll period between each device, in hours, minutes, seconds. Default is minutely, that is 00000h 01m 00s. Note this varies from the “standard” poll scheduler in drivers, where the poll period is the time to complete the entire poll cycle (all devices). See the “Actual Poll Period” property for statistics on the entire poll time.

- **Statistics Start**

Read-only timestamp of the start of all accumulated PAN info statistics.

- **Node Count**

Read-only count of both the “current” nodes and “average” nodes with polled PAN info.

- **Average Node Poll Time**

Time in milliseconds for average polled node.

- **Inter Node Delay**

Time in seconds between PAN info polling one device to another device.

- **Busy Time**
Percentage of PAN info polling time to total uptime of network.
- **Actual Poll Period**
Average time for PAN info polling to complete once for all devices.
- **Debug**
If set to true enables debug of PAN info polling; the default is false.

Note: Pan Info Poller has three available actions, described as follows:

- *Enable* — set the Enabled property to true.
- *Disable* — set the Enabled property to false.
- *Reset Statistics* — reset Pan Info Poller statistics (as above, including Statistics Start, Average Node Poll Time, Inter Node Delay, and so on).

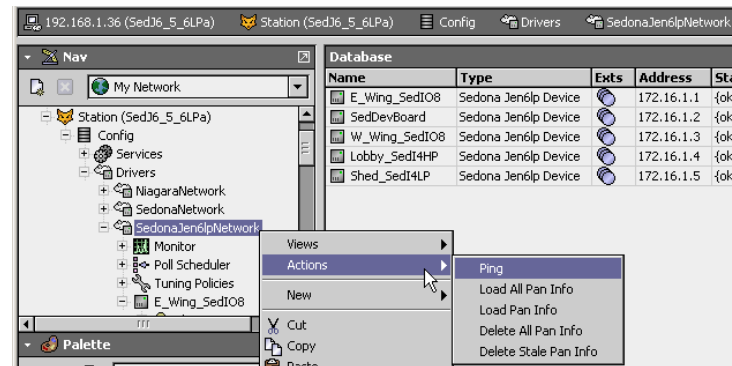
SedonaJen6lpNetwork Network Steady State Time property

Among [SedonaJen6lpNetwork properties](#) is this one, found near the bottom of the property sheet:

- **Network Steady State Time**
Allows specifying the time after station startup (say from a reboot) before the Sedona Jennic network is considered in a “steady state”. Until this timer expires, various service threads do not run, including device ping monitor, proxy point polling, PAN info polling, and automatic writes to proxy points (such as from tuning policy rules like “Write On Start” and “Write On Up”). This allows all devices to join the 6LoWPAN network before initiating application-level communications.
The default time is 1 minute (00000h 01m 00s). In cases of large and/or “deep” networks, adjusting upwards may prevent “write fault” or similar errors from occurring just after station startup.

SedonaJen6lpNetwork actions

Figure 3-11 SedonaJen6lpNetwork actions



In addition to the typical “Ping” action, the [SedonaJen6lpNetwork](#) has four actions related to “PAN info”. Actions on the network include the following:

- **Ping**
Verifies station communications with the coordinator (Sedona Jennic option card).
- **Load All Pan Info**
Forces a reload of PAN info from the coordinator and all devices with PAN info (routers).
- **Load Pan Info**
Forces a reload of PAN info from the coordinator only.
- **Delete All Pan Info**
Removes all station PAN info from the coordinator and all devices with PAN info (routers).
- **Delete Stale Pan Info**
Removes all stale station PAN info from the coordinator and all devices with stale PAN info (routers).

For related details, see:

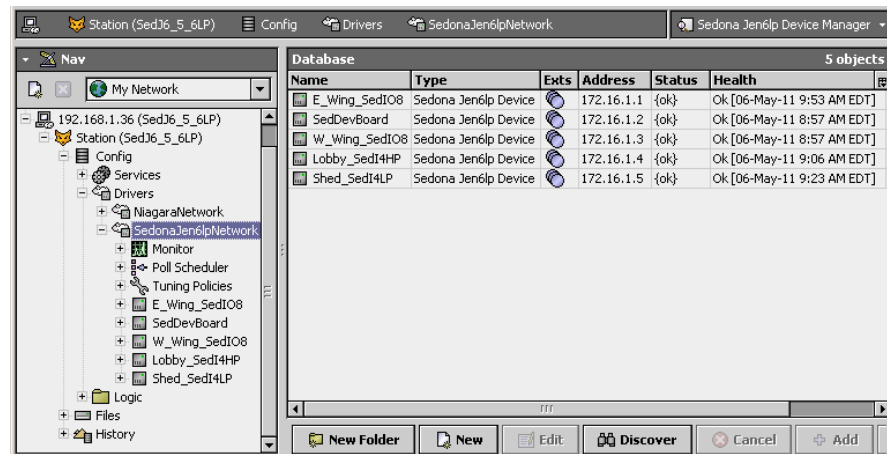
- [“SedonaJen6lpNetwork Pan Info properties”](#) on page 3-18
- [“Pan Sheet view of SedonaJen6lpNetwork”](#) on page 3-22

Note: Also refer to the Jennic Network Visualization (Pan Sheet) - Engineering Notes document.

Sedona Jen6lp Device Manager view

The **Sedona Jen6lp Device Manager** is the default view on a **SedonaJen6lpNetwork**, used to discover, add, and edit SedonaJen6lpDevice components that represent wireless Jennic-based devices.

Figure 3-12 Sedona Jen6lp Device Manager view

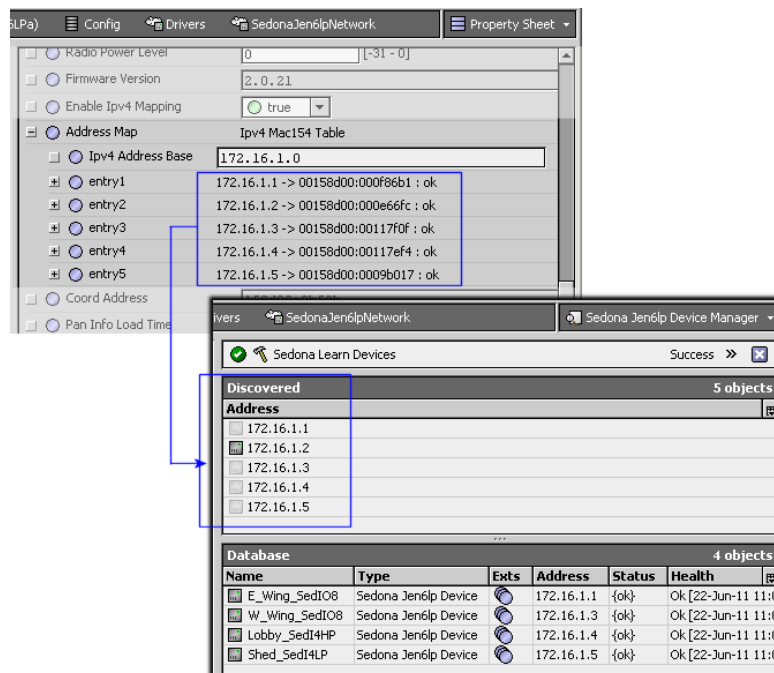


This view operates similar to the device manager in other NiagaraAX drivers that provide online discovery (Learn Mode)—see “About the Device Manager” in the *NiagaraAX Drivers Guide* for general information. Included are available “device folders” using the **New Folder** button, where each folder has its own **Sedona Jen6lp Device Manager** view. Also provided is the ability to discover and then **Match** a discovered device to an existing (and possibly replicated) SedonaJen6lpDevice component. Other details on the Sedona Jen6lp Device Manager are in the following sections:

- [Jen6lp Device Manager Learn Mode notes](#)
- [Jen6lp Device Manager data columns](#)

Jen6lp Device Manager Learn Mode notes

Figure 3-13 Address Map entries (of network) determine Discover results in Jen6lp Device Manager



When you click **Discover** to enter learn mode, and the “Sedona Learn Devices job” runs, it is different than most other drivers, as this does not initiate online messaging. Instead, the learn job reads the “Address Map” entries (mapped IPv4 addresses) of the SedonaJen6lpNetwork component, and for any listed as “ok” (Joined), they appear in the top Discovered pane of the view. See [Figure 3-13](#).

Typically you initially add device components by their default name, for example “dev 172.16.1.2”, and then later rename the devices going by your recorded list of MAC addresses that associates each to a particular known device (and installed location).

Note: For step-by-step details on using this view, see *“Discover and add SedonaJen6lpDevices”* on page 2-5.

Jen6lp Device Manager data columns

The Sedona Jen6lp Device Manager has a tabular **Database** pane at all times, and while in “Learn Mode”, above that a **Discovered** pane, as shown in Figure 3-13.

Currently the only data column in the **Discovered** pane is:

- **Address**
Remapped IPv4 address of each Jennic-based device, from the dynamically created entries under the SedonaJen6lpNetwork’s “Address Map” container slot.

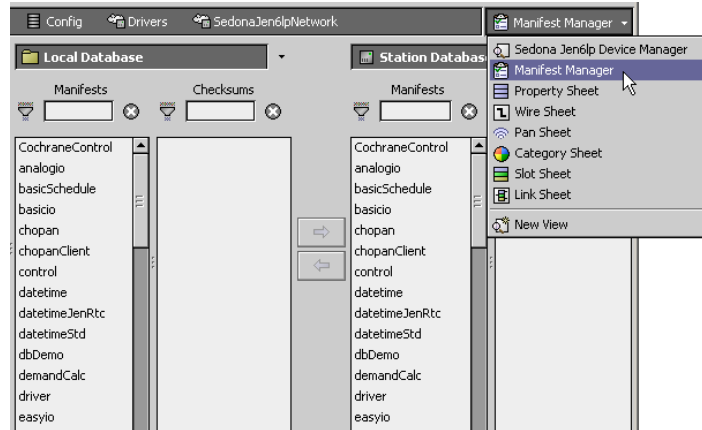
Table columns in the **Database** pane of the *Sedona Jen6lp Device Manager* (by default) include the following:

- **Name**
Niagara name for the device. Default is “dev IPv4 address”, for example: dev 198.162.1.4
- **Type**
Always “Sedona Jen6lp Device” (cannot modify).
- **Exts**
Points extension for the device — double-click for its **Sedona Point Manager** view.
- **Address**
Remapped IPv4 address of the Jennic-based device.
- **Status**
Device component status, typically “ok” but possibly “fault” (license or other issue) or “down”.
- **Health**
“Ok” or “Fail” with timestamp of last device monitor ping. Fail typically coincides with a down status.
- **Fault Cause**
Typically blank, unless health is Fail, where any available “Fault Cause” string appears.

Note: Not included by default, but available in the *Table Options (menu)* of the Database pane, are items “Enabled” and “Credentials”.

Manifest Manager view of SedonaJen6lpNetwork

Figure 3-14 Manifest Manager view is one available view on the SedonaJen6lpNetwork



As shown in Figure 3-5, the **Manifest Manager** is an available view on a *SedonaJen6lpNetwork*—note it is the *same view* that is available on a *SedonaNetwork*.

You typically use this view to copy Sedona kit manifest files (“manifests”) from your Workbench host (Local Database) to the JACE hosting the station (Station Database). In this case, you copy manifests *from the left side to the right side*. The station running any Sedona Framework network needs direct access to all kit manifest files for any networked device, to support Sedona proxy point discovery and point updates. For a related procedure, see *“Run the Manifest Manager to ensure kit manifests are loaded”* on page 2-8.

However, this view also works “bi-directionally”. That is, you can use it to copy manifest *from the JACE* running the SedonaJen6lpNetwork *to your Workbench computer*, copying from “right to left”. This may be useful if you access the station using a different Workbench host than was used to engineer the station. Note that Workbench requires direct access to the manifests for all kits in a Sedona Framework device when making a Sox connection to it (typically, a Sox tunneling connection)—otherwise, the connection will fail with an exception (exception error “details” usually list the missing manifest file or files).

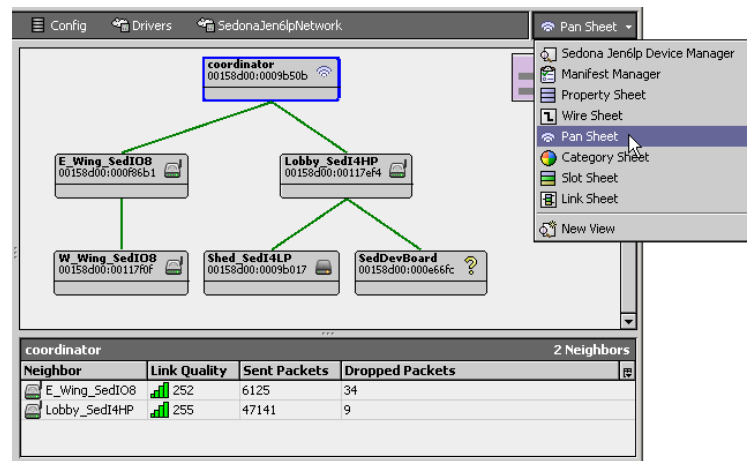
Note: *Niagara Workbench with Sedona Framework TXS also provides a similar “upper level” **Sedona Manifest Manager** view, launched from the menu bar: **Tools > Sedona Manifest Manager**. In that view, the left pane is “Web Database” (source, read-only), while the right (target) pane is your Workbench computer. Providing that you have Internet connectivity, this gives access to manifest files available at sedonadev.org.*

Find complete details about using the Manifest Manager in the *Sedona Framework Manifest Manager - Engineering Notes* document.

Pan Sheet view of SedonaJen6lpNetwork

As shown in Figure 3-5, the **Pan Sheet** is an available view on a [SedonaJen6lpNetwork](#)—note this view is unique to the SedonaJen6lpNetwork.

Figure 3-15 Pan Sheet is one available view on the SedonaJen6lpNetwork



This view is a diagnostic tool that can *graphically show* the “self healing tree structure” of the underlying JenNet wireless network, along with associated “PAN info” statistics stored in the (JACE) coordinator and any devices operating as “router” nodes. Included in statistics is the radio “Link Quality”, representing strength of last packet received. The (JACE) coordinator node is always at the top of the network tree.

Nodes in the view’s top pane are selectable, with context-sensitive PAN info data about neighbor nodes (children and/or parent) shown in the table at the view bottom. Data from router nodes is retrieved by the “PanInfo Poller” mechanism of the SedonaJen6lpNetwork (once enabled), or by explicit “Load Pan Info” actions. Related PAN info actions are on the network component, as well as any SedonaJen6lpDevice component.

Note: *Usage requires installing one or more Sedona Framework kits in router-capable devices (only), and some minimal app configuration in those devices. Additionally, in the station each SedonaJen6lpDevice should have its “Device Type” property set (from the default “Unknown”) to either “Router” (if a router-capable device) or “End Device” if not capable of router operation.*

Note that PAN info polling is not recommended for continuous operation, as it consumes network bandwidth. However, you can still use the Pan Sheet without polling, using the “Load All Pan Info” action on the SedonaJen6lpNetwork. In general, Pan Sheet usage is helpful for initial network setup and/or troubleshooting communications problems.

Find complete details about using the **Pan Sheet** in the *Jennic Network Visualization (Pan Sheet) - Engineering Notes* document.

For other related sections in *this* document, see:

- “[SedonaJen6lpNetwork Pan Info properties](#)” on page 3-18
- “[SedonaJen6lpNetwork actions](#)” on page 3-19

About the SedonaJen6lpDevice component

The SedonaJen6lpDevice is the device-level component representing a wireless Jennic-based device, and is a direct child of a [SedonaJen6lpNetwork](#). It contains properties and slots typical to most driver's device components—see “Common device components” in the *Drivers Guide* for general information on device component status properties, health, and alarm source info.

No special views are available on the SedonaJen6lpDevice, apart from the standard ones (Property Sheet, Wire Sheet, Category Sheet, Slot Sheet, Link Sheet). However, it has a standard **Points** device extension with **Sedona Point Manager**, and a **Chopan Virtual** gateway—unique to this device type.

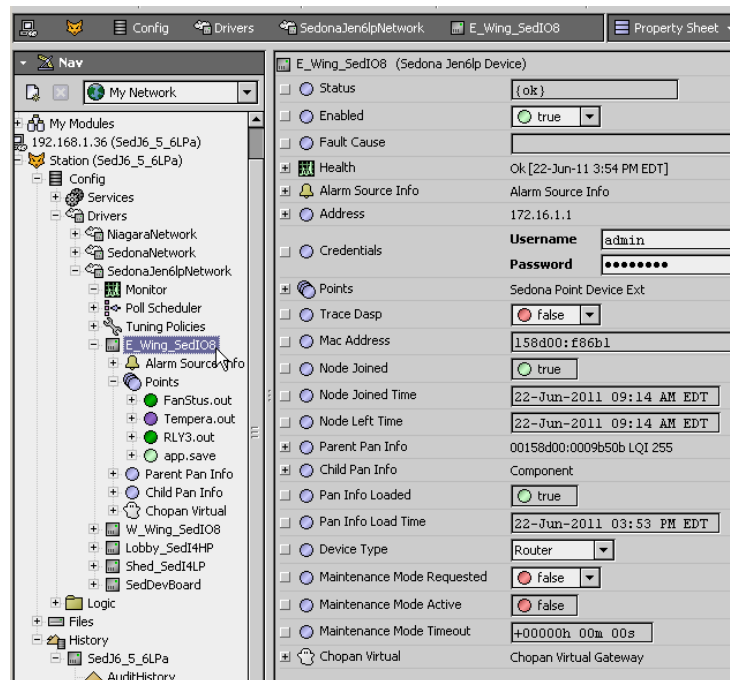
The following sections provide more details:

- “[SedonaJen6lpDevice properties](#)” on page 3-23
- “[SedonaJen6lpDevice actions](#)” on page 3-25
 - “[About maintenance mode](#)” on page 3-26
- “[About the Chopan Virtual gateway](#)” on page 3-26

SedonaJen6lpDevice properties

Figure 3-16 shows the property sheet view of a SedonaJen6lpDevice—its default view.

Figure 3-16 SedonaJen6lpDevice property sheet



Properties and container slots unique to the SedonaJen6lpDevice include:

- **Address**
 - **Ip** — Remapped IPv4 private network address of the Jennic-based device, automatically assigned by the coordinator (under the “Address Map” container slot of SedonaJen6lpNetwork).
 - **Sox Port** — The port the device's Sox service is listening on. Default port is 1876.
 - **Chopan Port** — The port the device's CHoPAN server is listening on. Default port is 1810.

Note: If you know that a Jennic-based device is using a different Sox port and/or Chopan port, change it to match in the address property.


- **Credentials**
The credentials used to authenticate to the device in a Sox connection.
- **Points**
The single device extension, with a default **Sedona Point Manager** view. For related details, see “[About the Sedona Point Manager](#)” on page 3-33.
- **Trace Dasp**
If enabled, and you have station logSetup (in spy) set to “trace” for SedonaJen6lpNetwork.sox, this directs extra device-specific messaging of the Sox traffic over the DASP (Datagram Authenticated Session Protocol) to the station's output (visible in platform Application Director).

- **Mac Address**
Read-only. The unique 802.15.4 MAC address of the device, in hexadecimal notation without leading zeroes, for example: 158d00:9b017.
- **Node Joined**
Read-only. Indicates if the node has joined the Jennic wireless network.
- **Node Joined Time**
Read-only. Timestamp of when the node last joined the Jennic wireless network.
- **Node Left Time**
Read-only. Timestamp of when the node last left the Jennic wireless network.
- **Parent Pan Info**
Container SedonaJen6lpNeighborEntry component for the *single parent node* for the device, 802.15.4MacAddress LQI n (for example, 00158d00_0009b50b LQI 255)
Contains read-only JenNet PAN info about the parent node, including:
 - Is Sleeping End Device — Always `false`. (Whether device is a battery powered (hibernating) end device type (`true`) or a continuously powered device (`false`)).
 - Link Quality — Value between 0-255, representing strength of last packet received from this node. Values over 60 represent a good link.
 - Packets Lost — Number of packets sent to this node for which an ack(nowledgement) was not received. Note an ack is not expected on all packets.
 - Packets Sent — Number of packets sent to node from this device.
 - Packets Received — Number of packets device has received from this node.
 - Mac154 — The unique 802.15.4 MAC address of this node, in hexadecimal notation without leading zeroes, for example: 158d00:9b50b
 - Stale — Typically false, else true if this node entry is stale, meaning the device represented is no longer the parent. For example, if a device resets and rejoins the JenNet network, it may have a new parent (the coordinator or another router node). In this case, this Stale values shows true.
- **Child Pan Info**
Container for one or more dynamically-added SedonaJen6lpNeighborEntry child components, if applicable. Each contains read-only JenNet PAN info about a direct child node.
Note: *If the device has no child nodes, this container is empty.*
 - MAC802.15.4MacAddress LQI n (for example, MAC00158d00_0009b017 LQI 186)
 - Is Sleeping End Device — Whether device is a battery powered (hibernating) end device type (`true`) or a continuously powered device (`false`).
 - Link Quality — Value between 0-255, representing strength of last packet received from this node. Values over 60 represent a good link.
 - Packets Lost — Number of packets sent to node for which an ack(nowledgement) was not received. Note an ack is not expected on all packets.
 - Packets Sent — Number of packets sent to node from this device.
 - Packets Received — Number of packets this device has received from node.
 - Mac154 — The unique 802.15.4 MAC address of the node, in hexadecimal notation without leading zeroes, for example: 158d00:9b017
 - Stale — Typically false, else true if this node entry is stale, meaning the node represented is no longer a child. For example, if a device resets and rejoins the JenNet network, it may have a new parent (say the coordinator or another router node). In this case, this Stale values shows true.
- **Pan Info Loaded**
Read-only indication if JenNet PAN info for this device is available (`true`) or if it has been deleted (`false`), perhaps from a device action.
- **Pan Info Load Time**
Read-only timestamp of when PAN info last loaded, say from a device action, or from the ongoing PAN info poller mechanism (from coordinator/SedonaJen6lpNetwork).
- **Device Type**
Configuration property that specifies whether the Jennic-based device is either “router capable” (Router), or reduced function “end device” (End Device—perhaps even a hibernating device), or simply “Unknown”. The default is Unknown.
Note: *Set this from the default (Unknown) to one of the two other selections (Router or End Device), depending on the Jennic capability of the device. This is particularly important to allow “PAN info” polling and related data to be used in the Pan Sheet view of the SedonaJen6lpNetwork.*
- **Maintenance Mode Requested**
(Applies to hibernating devices only) This provides indication if a “maintenance mode request” has been given on this device (`true`) or not (`false`). The “Request Maintenance Mode” *action* on the

device is typically how a request is given. This value goes to true when the action is invoked, then to false when maintenance mode becomes active—when in Workbench a modal popup “maintenance mode active” appears. See “[SedonaJen6lpDevice actions](#)” on page 3-25.

Note: *Maintenance mode only applies to a device that hibernates (say a battery-powered device), and requires Chopan usage in the device’s app. If the device does not hibernate, you can safely hide this property, as well as the other two Maintenance Mode properties (and also the “Request Maintenance Mode” action), working from the SedonaJen6lpDevice’s slot sheet.*

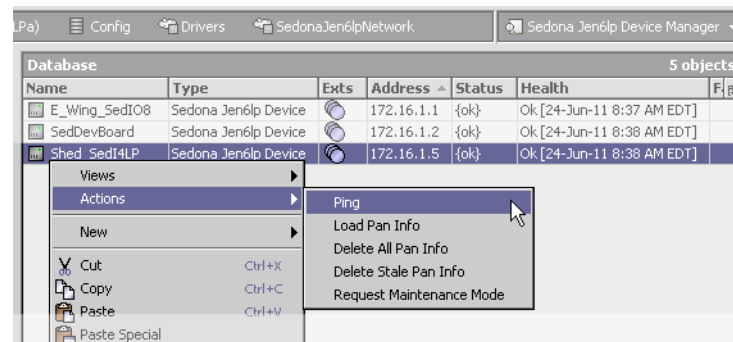
Currently, Sedona Framework support for hibernating devices is not widely available, so this is another possible reason to hide these Maintenance Mode properties and action.

- **Maintenance Mode Active**
(Applies to hibernating devices only) Provides indication if “maintenance mode” is currently active on this device (true) or not (false). It becomes true only after the modal “maintenance mode active” popup appears in Workbench, and returns to false when the maintenance mode time out period (configured in the device’s Sedona Framework app) expires. During this period, the device is not hibernating, which enables it to accept a Sox connection from Workbench. Also see the [Note](#) on the “Maintenance Mode Requested” property.
- **Maintenance Mode Timeout**
(Applies to hibernating devices only) Read-only indication of the current maintenance mode time-out period, as configured in the Sedona Framework app of the device. This specifies the duration of the maintenance mode active period. If a Sox connection is not established during an active period, the device returns to normal hibernation operation, until the next maintenance mode request. Also see the [Note](#) on the “Maintenance Mode Requested” property.
-  **Chopan Virtual**
A “virtual gateway” used to configure “Chopan points” in the device’s Sedona Framework app, via a virtual “Chopan Network” with “Chopan Devices” and “Chopan Points” in the Niagara station. See “[About the Chopan Virtual gateway](#)” on page 3-26.

SedonaJen6lpDevice actions

[Figure 3-17](#) shows the default actions available on a SedonaJen6lpDevice.

Figure 3-17 SedonaJen6lpDevice actions (default)



Device actions include the following:

- **Ping**
Verifies join status with the coordinator (SedonaJen6lpNetwork).
- **Load Pan Info**
Forces a load of the device’s JenNet PAN info into its Parent Pan Info and Child Pan Info properties.
- **Delete Pan Info**
Deletes all JenNet PAN info from the device’s Parent Pan Info and Child Pan Info properties.
- **Delete Stale Pan Info**
Deletes all JenNet PAN info entries currently marked “stale” (property Stale = true). These entries may exist after the tree structure of the Jennic wireless network changes.
- **Request Maintenance Mode**
Applies only to hibernating devices (say battery-powered)—to provide the ability to make a Sox connection to it from Workbench. For more details, see “[About maintenance mode](#)”.
Note: *If a non-hibernating device, this action has no purpose and may be safely hidden. As currently Sedona Framework support for hibernating devices is not widely distributed, this is yet another reason to hide this action.*

About maintenance mode

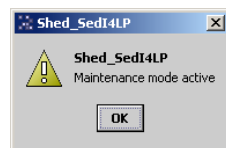
Note: *Maintenance mode is a routine that applies to hibernating devices only. At the time of this document, Sedona Framework support for hibernating devices is not widely available. However, NiagaraAX support for such devices, including “Maintenance Mode” operation, is included in the SedonaJen6lpNetwork driver. Most hibernating devices, typically battery-powered devices, remain in a low-power or “hibernating” state. During this state, the Sedona VM and RF communications are disabled to conserve battery life. Consequentially, Workbench cannot make a Sox connection to the device.*

In order to allow a Sox connection from Workbench to a device that hibernates, its Sedona Framework app is configured with a “ChopanNetwork” *client* that periodically queries the coordinator (JACE) to see if it should enter a *maintenance mode*.

This request is typically done using the “Request Maintenance Mode” action on the SedonaJen6lpDevice that represents the hibernating device. Once invoked, upon the next “maintenance mode check” message from the device (as part of its Chopan setup), the device sees that flag set, and then suspends hibernation for a “maintenance mode timeout” period.

At that time, in Workbench a modal “Maintenance mode active” popup appears, with the Niagara name of the associated SedonaJen6lpDevice component. See [Figure 3-18](#) for an example popup.

Figure 3-18 Example maintenance mode popup in Workbench (for device named Shed_SedI4LP)



The popup signals the start of the “maintenance mode active time”. During this period, before the timeout, you can use Workbench to open a Sox connection to that device (typically a Sox tunnel connection). If a Workbench Sox connection is not made in that time period, the device returns to hibernating, that is, resumes its normal operation.

Three related properties of a SedonaJen6lpDevice indicate maintenance mode states and the (Sedona configured) maintenance mode timeout duration. See “[SedonaJen6lpDevice properties](#)” on page 3-23.

About the Chopan Virtual gateway

The Chopan Virtual gateway is used to configure “Chopan points” in the device’s Sedona Framework app, via a virtual “Chopan Network” with “Chopan Devices” and “Chopan Points” in the Niagara station. The resulting Chopan components added in the device’s app act as the Chopan *client interface* to other devices’ “Chopan servers” (including the one for the station, in the SedonaJen6lpNetwork).

To support virtual gateway access from Niagara, the Jennic-based device requires the chopan kit installed, and its Sedona Framework app configured with components from it.

- If a non-hibernating (typically continuously-powered) device, Chopan usage is optional. However, Chopan as a “comm type” for normal Sedona proxy points offers bandwidth efficiencies. Also, via Chopan points, a device can exchange data directly with another networked Jennic-based device.
- Such configuration is *required* if a hibernating¹ device (typically a battery-powered device), as Chopan points must be used instead of normal Sedona proxy points to exchange data with the station. Chopan is also used in the “maintenance mode” mechanism for a hibernating device.

A brief overview on Chopan points follows in this document, in the following sections:

- [Required configuration in the Jennic-based device](#)
- [Walk-through using Chopan Virtual gateway](#)
- [Additional Chopan usage topics](#)

Note: *For complete details see the Sedona Framework Chopan Usage document.*

Required configuration in the Jennic-based device

To support Chopan, the Jennic-based device requires the following:

- The chopan kit installed.

¹ Currently, Sedona Framework support for hibernating devices is not widely available. Chopan Virtual gateway usage without this support should be minimal, as it is necessary only if a Jennic-based device needs to request data directly from another networked Jennic-based device.

- To function only as a CHoPAN client (hibernating device support), its Sedona Framework app must be configured with the following:
 - ChopanService (from chopan kit) added to its Service container.
 - ChopanNetwork (from chopan kit) added to its Service container.
 ChopanDevices and ChopanPoints can then be added to its app using the special manager views under the Chopan Virtual gateway of the SedonaJen6lpDevice that represent it in the station.
- To function as a CHoPAN server, its Sedona Framework app must be configured with the:
 - ChopanService (if not already present, from chopan kit) added to its Service container.
 - ChopanServlet (if not already present, from chopan kit) added under the ChopanService.

Chopan server configuration is typical for most Jennic-based devices (non-hibernating), to allow Sedona proxy point updates using a tuning policy with “Chopan” as the comm type. See [“Chopan comm type considerations”](#) on page 3-15. Additionally, the device can be configured for Chopan client operation too, if direct access to data in some other networked Jennic-based device is needed.

Walk-through using Chopan Virtual gateway

This walk-through describes using CHoPAN to exchange data between a hibernating Jennic-based device and the JACE station, where the “Chopan Server” of the SedonaJen6lpNetwork is enabled (default), and the hibernating device has its app configured as a CHoPAN client and server (as described in section [“Required configuration in the Jennic-based device”](#) on page 3-26). Note all of the following *replaces* Sedona proxy point usage for this hibernating Jennic-based device.

Example Chopan point setup using Chopan Virtual gateway

Before using the Jen6lpDevice’s Chopan Virtual gateway, it would be best to have “target” writable Niagara points ready to receive read-only values from the hibernating Jennic-based device. In the Niagara jen6lp palette, there is a folder named “ChopanTargetPoints”, with two conveniently “trimmed” writable points:

- ChBoolTarget — a normal BooleanWritable, but with *all actions and all inputs except “in10” hidden*.
- ChFltTarget — a normal NumericWritable, but with *all actions and all inputs except “in10” hidden*.

Actions and other input slots are hidden to avoid confusion trying to control (write to) this read-only value from Sedona, say if dragging this control point on a Px page. Typical usage of these points is to receive some value like a temperature, or an equipment state from the device’s Sedona Framework app.

As a best practice, and for database portability/replication, it is recommended that you place such Chopan target points in a *folder under the SedonaJen6lpDevice* that represents the hibernating device. Note that the standard “Points” folder is not the best choice—as these are *not* proxy points, and for this reason do not appear in the Sedona Point Manager view.

Figure 3-19 Adding ChopanTargetPoints for read-only values, in folder under device component

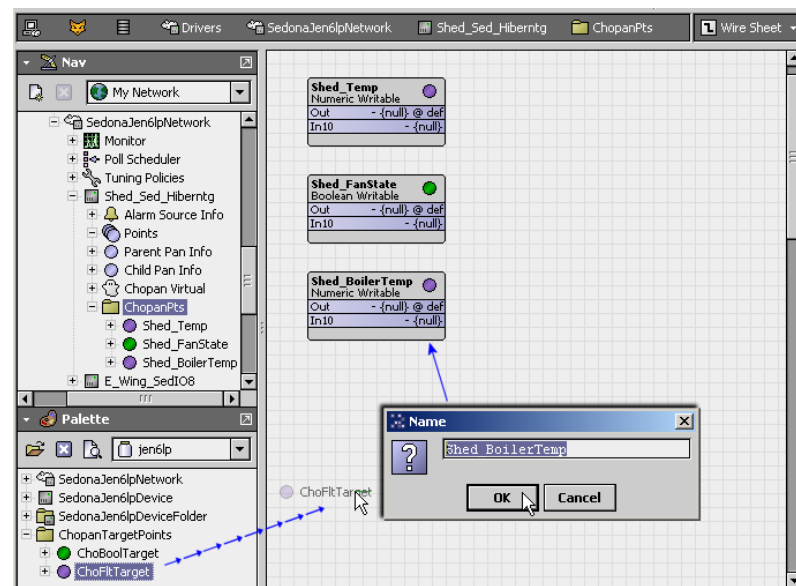


Figure 3-19 above shows three such points copied from the jen6lp palette into a “ChopanPts” folder under the SedonaJen6lpDevice (named “Shed_Sed_Hibernatng”).

Now, onto using the gateway.

Note: In the following steps, virtual components must be clicked (versus expanded) to “enter” them.



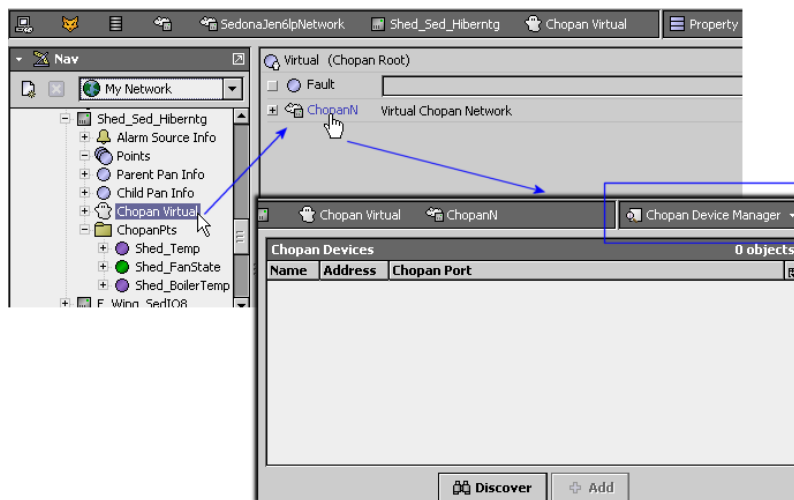
- Step 1 The  **Chopan Virtual** gateway is clicked, and it soon expands to reveal a  **ChopanN** component (Virtual Chopan Network) in its property sheet.
- Step 2 The ChopanN component is clicked, and the view changes to the **Chopan Device Manager** for that device. See Figure 3-20.

Figure 3-20 Chopan Device Manager is view on the ChopanN (Virtual Chopan Network)



As shown in Figure 3-20, initially the **Chopan Device Manager** is empty—a Discover is needed to find other devices configured with a Chopan server.


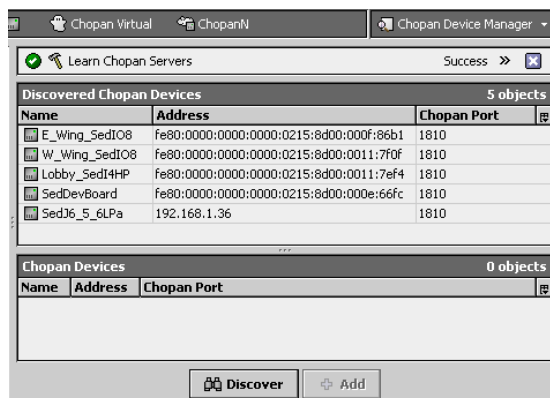
- Step 3 The  **Discover** button is clicked, launching a **Learn Chopan Servers** job.

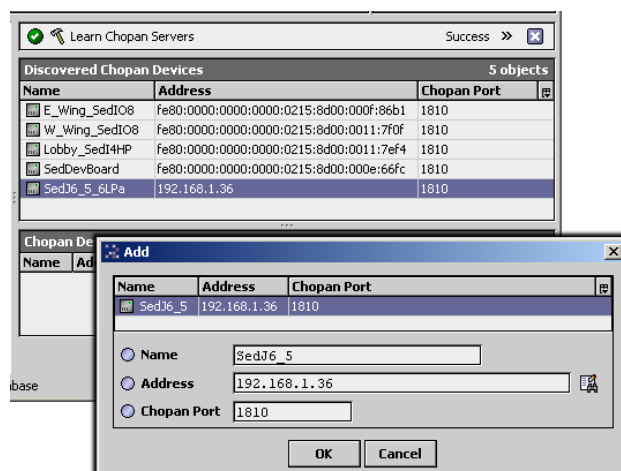
Figure 3-21 Discovered Chopan servers in Chopan Device Manager



As shown in Figure 3-21, five devices were found, of which one is the JACE station (“SedJ6_5_6LPa”). In most cases, this is the device of primary interest—and the one to be added here.

- Step 4 The discovered JACE station is double-clicked to bring up the Add dialog, shown below.

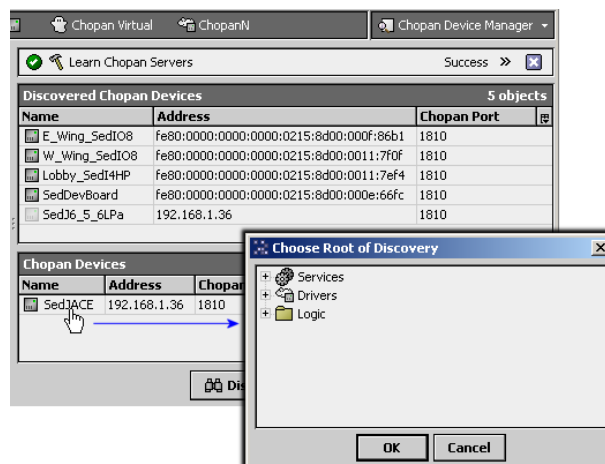
Figure 3-22 Adding Chopan Device that represents the JACE station



Notice that Name is truncated at 7 characters—the maximum number of characters for naming any Sedona component. In this case, the name is changed from “SedJ6_5” to “SedJACE”, and then added (OK).

Step 5 The added SedJACE Chopan Device is then double-clicked.

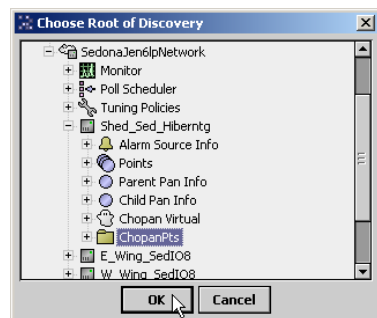
Figure 3-23 Point discovery of Chopan Device brings up



As shown in Figure 3-23, this brings up a “Choose Root of Discovery” dialog, reflecting the contents of the **Config** node of the JACE station, in an expandable tree.

Step 6 The tree is expanded to find the previously added “ChopanTargetPoints” for this device.

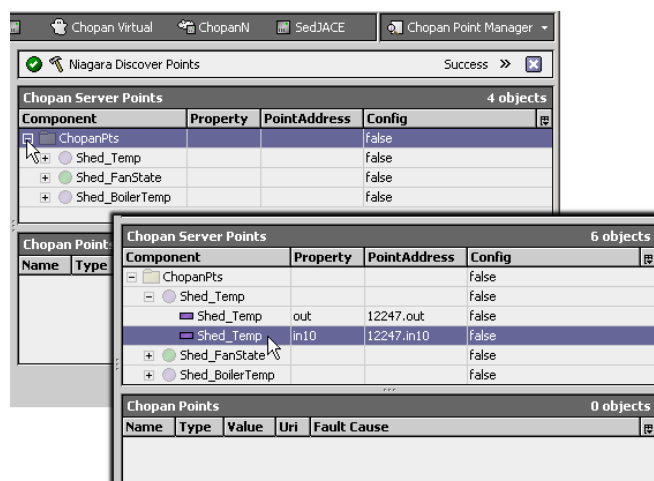
Figure 3-24 Drivers leaf expanded to find Chopan target points for this device



As shown in Figure 3-24, the previously-created folder “ChopanPts” is selected as the root (OK). The view changes to the **Chopan Point Manager** for the (JACE) device.

Step 7 In the Discovered pane, the folder is expanded to reveal the target points, then available properties.

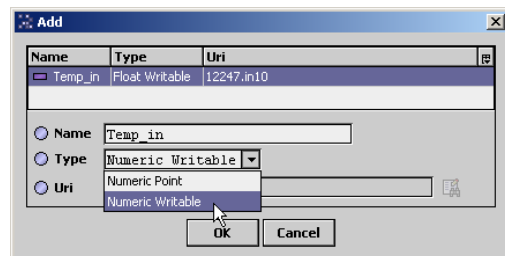
Figure 3-25 Expanding root folder in Discover to find properties of Chopan target points.



The “in10” property is double-clicked to bring up the **Add** dialog.

Step 8 The **Add** dialog defaults to a NumericPoint, with truncated name (in this case, “Shed_Te”).

Figure 3-26 Add dialog for Chopan target point, changing to writable type point

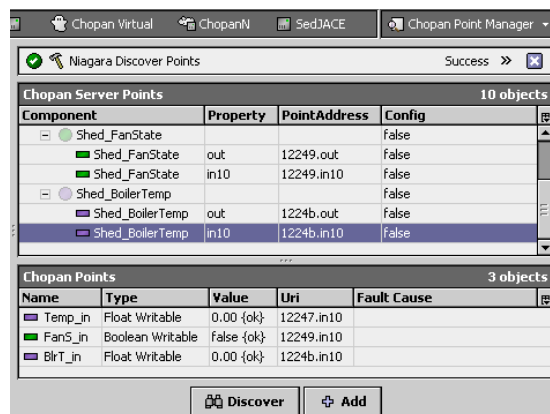


As shown in [Figure 3-26](#), name is edited to “Temp_in”, and point type *changed* to NumericWritable, and the point is added (**OK**). The actual Sedona component added is a ChoFtWr (Chopan Float Writable).

This needs to be a writable point, because the source Sedona temperature component will be linked to it in the device’s Sedona Framework app.

Step 9 The same method is used to add two other Chopan points in the device, also writable ([Figure 3-27](#)).

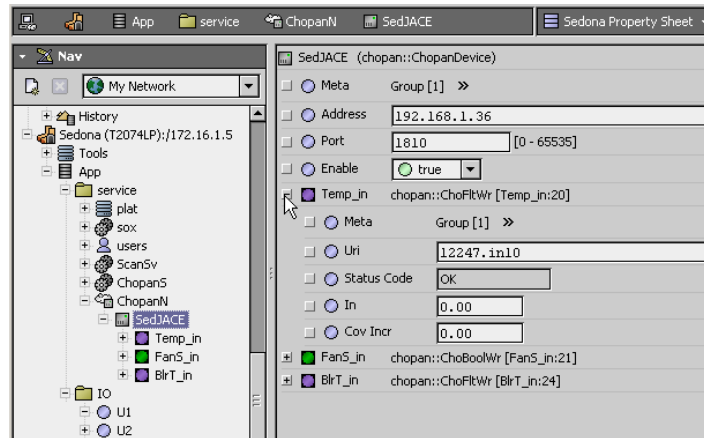
Figure 3-27 Three Chopan points now in the Chopan Point Manager view



Note that these three Chopan point components, as well as the parent Chopan device component, *now exist in the Sedona Framework app* of the hibernating device. To make them operational, a Sox Workbench connection must be made to this device, such that the source Sedona components can be linked to them.

Step 10 A “Maintenance Mode Request” action is invoked on the SedonaJen6lpDevice for this device, and following the “Maintenance Mode Active” popup, a tunneled Sox Workbench connection is made to it.

Figure 3-28 Workbench Sox connection made to hibernating device, showing added Chopan points



As shown in Figure 3-28, the Chopan points in the device's app are under the "ChopanN", "*ChopanDeviceName*" node under the **service** folder (and must remain there). One way to link the source Sedona components to them is using the Nav tree, and right-click "Link Mark" and "Link From" menu options.

- Step 11 Source Sedona components are linked to the Chopan points, using "Link Mark" and "Link From" methods.

Figure 3-29 Linking source Sedona components to Chopan writable points

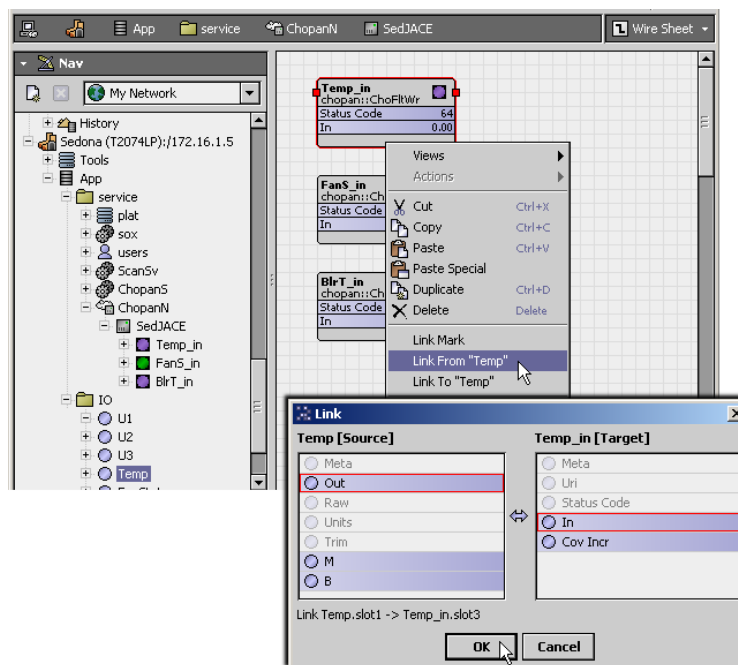


Figure 3-29 shows the **Link** dialog for the "Temp" (erature) source component to the "Temp_in" writable Chopan point, where the "In" slot is used.

- Step 12 After making source links to all the writable Chopan points, the app must be saved.

Figure 3-30 Saving Sedona Framework app of the device after linking in Chopan points

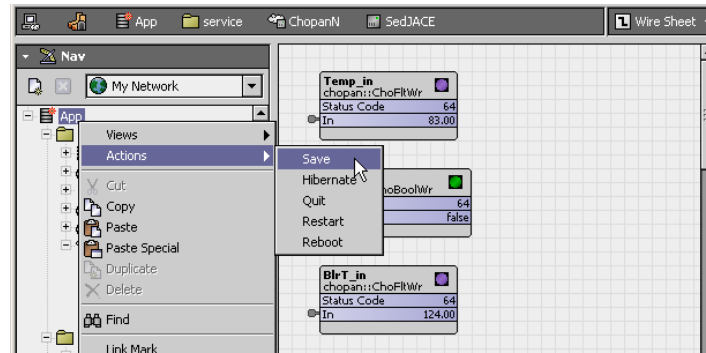
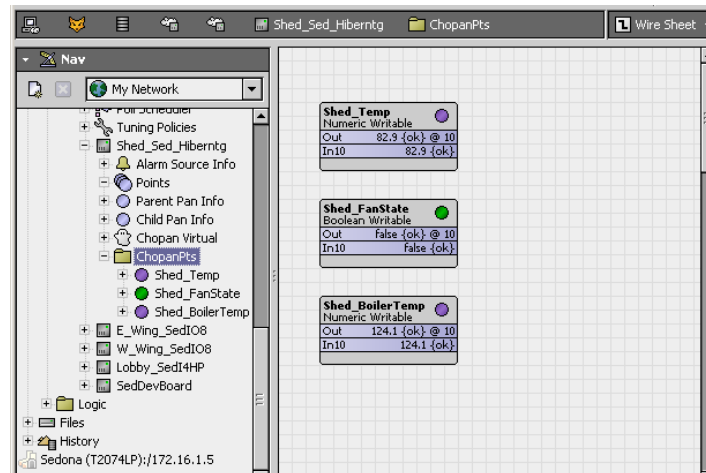


Figure 3-30 shows the right-click **Save** command invoked on the App node. The Workbench Sox session to the hibernating device is now closed, and the target Chopan points are checked in the JACE station.

Step 13 Chopan target points in the station are now receiving updates from the hibernating device, via CHoPAN, as shown in Figure 3-31.

Figure 3-31 Chopan target points in station, with values received via CHoPAN.



Control points shown in Figure 3-31 still need point facets assigned, and whatever linkage into station control logic, and/or bindings to Px page(s).

Additional Chopan usage topics

The following topics are not in the previous Chopan point “walk-through”, but are mentioned briefly below. For more details on Chopan topics, refer to the *Sedona Framework Chopan Usage* document.

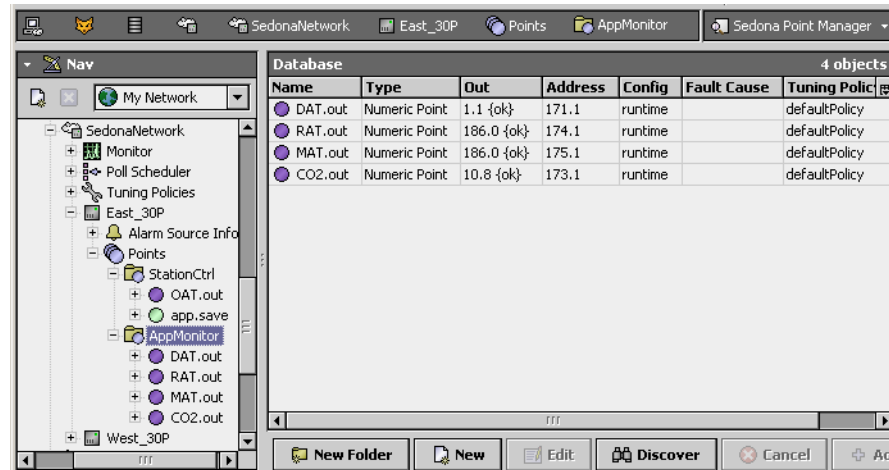
- When the Chopan Server of the SedonaJen6lpNetwork is enabled, this automatically results in the creation of a station user named “CHoPAN”. This is necessary because CHoPAN is an unauthenticated protocol, where CHoPAN servers do not authenticate or prevent and reads or writes to local components. Niagara stations allow reads of any appropriate component, but *writes* to station components are checked for authorization (as part of this CHoPAN user scheme). Therefore, any incoming CHoPAN requests that result in station database modification are checked against the permissions assigned to the CHoPAN user. In other words, the Niagara component being written must be assigned to a category for which the CHoPAN user has write privileges.
- Any Sedona Framework Jennic-based device can be configured in its app to support “service pin notifications”, using components from the chopan kit. The app must contain the “ChopanS” (Chopan-Service) and “ChopanN” (ChopanNetwork) components, with others optional. See the “Service Pin support” section in the *Sedona Framework Chopan Usage* document.
- Chopan points have a *status code* slot showing a numerical value that reflects the response of the latest request. These codes can be made visible in the Chopan Point Manager view of a Chopan server device (under the SedonaJen6lpDevice’s **Chopan Virtual** gateway). Status codes can be useful for troubleshooting purposes. Refer to the “Chopan Diagnostics” section in the *Sedona Framework Chopan Usage* document for status code values and their meanings.

About the Sedona Point Manager

The **Sedona Point Manager** is the default view on the **Points** device extension (or a Points folder under that) for any type of Sedona device component (SedonaDevice or SedonaJen6lpDevice), and operates the same way for either type of device.

Note: For any hibernating *SedonaJen6lpDevice* (typically a battery-powered device), use of this view to create *Sedona proxy points* is not recommended. Instead, use the device's *Chopan Virtual gateway* to integrate data with the station. See [“About the Chopan Virtual gateway”](#) on page 3-26 for an overview.

Figure 3-32 Sedona Point Manager example



The **Sedona Point Manager** is similar to the point managers in other NiagaraAX drivers that feature points folders and online point discovery (Learn mode). For general information on point managers, refer to the *Drivers Guide* sections “Points New Folder and New” and “About Point Discover, Add and Match (Learn Process)”.

The following sections provide more details about the **Sedona Point Manager**:

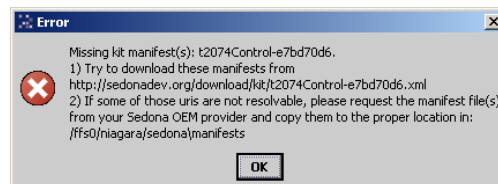
- [“Unique things about the Sedona Point Manager”](#) on page 3-33
- [“Sedona Point Manager “Discovered” notes”](#) on page 3-34
- [“Add dialogs in Sedona Point Manager”](#) on page 3-35
- [“Sedona Point Manager “Database” notes”](#) on page 3-37

Unique things about the Sedona Point Manager

The following things are unique to the **Sedona Point Manager**:

- **Manifests required on the JACE**
For proxy point support of any Sedona Framework device, the host (e.g. JACE) running the station with the Sedona Framework network requires the *manifest* file for each kit installed in the device. Otherwise, a “Sedona Discover Points” job will fail, and an error popup similar to [Figure 3-33](#) below appears in Workbench.

Figure 3-33 Example error popup on Sedona Discover Points job (missing manifest)



To fix this, obtain the appropriate kit manifest file(s), and then transfer them to the JACE using the **Manifest Manager** view on either Sedona Framework network (SedonaNetwork, SedonaJen6lpNetwork). See [“Run the Manifest Manager to ensure kit manifests are loaded”](#) on page 2-8. Then rerun the point discovery.

For other related details in this document, see [“Manifest Manager view of SedonaNetwork”](#) on page 3-9 or [“Manifest Manager view of SedonaJen6lpNetwork”](#) on page 3-21. Find complete details about using the Manifest Manager in the *Sedona Framework Manifest Manager - Engineering Notes* document.

- **Action points available**
In addition to Sedona proxy points, the Sedona Point Manager lets you add “action points”, which are unique to Sedona Framework devices. Each action point is a primitive component that represents a single action on the source Sedona component. Action points do not read data values nor accept point extensions like alarm, history, and so on—and have only a single property (Address). An action point makes that same action available in Niagara, named using the action slot name on the parent Sedona component. If needed, you can add a display name for that action, working from the slot sheet of the action point. For more details, see [“About Sedona action points”](#) on page 3-41.
- **Default point type may need changing**
Each Sedona proxy point represents a single property of a Sedona component in the device’s Sedona Framework app. Knowledge of the device’s app is often required when making these decisions. Variations between data types in Sedona and Niagara should be understood for type selection of proxy points. There is no “enum” (multistate) data type¹ in Sedona, however, “enum” point types default for any selected property that uses an integer data type (int, long). Sometimes this may be appropriate. Other times, you may wish to change type when adding, for example from “Enum” to “Numeric” for an integer property representing a count.
For more details, see [“About Sedona proxy points”](#) on page 3-38, and [“Sedona and Niagara data types and null notes”](#) on page 3-39.

Sedona Point Manager "Discovered" notes


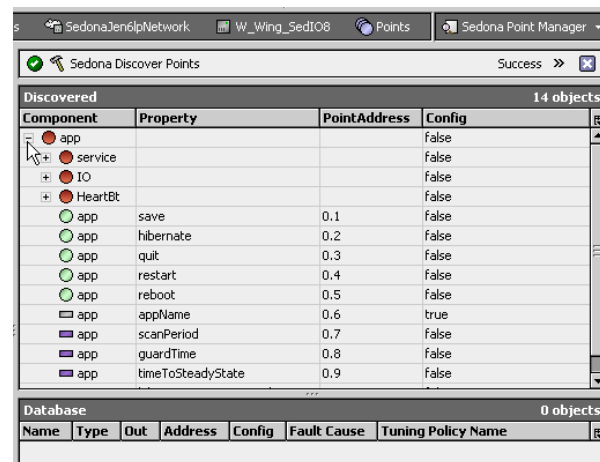




By default, the “Discovered” table for Sedona points shows a single, red icon,  **app** component at top.

Figure 3-34 Discovered Sedona components, folders, and child properties and actions



| Component | Property | PointAddress | Config |
|-----------|-------------------|--------------|--------|
| app | | | false |
| service | | | false |
| IO | | | false |
| HeartBit | | | false |
| app | save | 0.1 | false |
| app | hibernate | 0.2 | false |
| app | quit | 0.3 | false |
| app | restart | 0.4 | false |
| app | reboot | 0.5 | false |
| app | appName | 0.6 | true |
| app | scanPeriod | 0.7 | false |
| app | guardTime | 0.8 | false |
| app | timeToSteadyState | 0.9 | false |

When expanded, other components and folders appear with the same red icon—each is expandable, but not directly selectable. Only child *properties* ( ,  , ) or *actions* () are selectable. Note that Sedona component names cannot be more than 7 characters—as seen in the “Component” column.

Click to expand any component or folder, as shown in [Figure 3-35](#).

1. The boolean data type in Sedona is effectively multistate, as it can be either false, true, or null. This is the only multistate instance in Sedona. For more details, see [“Booleans as Enums”](#) on page 3-40.

Figure 3-35 Click plus (+) icon beside any discovered component to see all properties

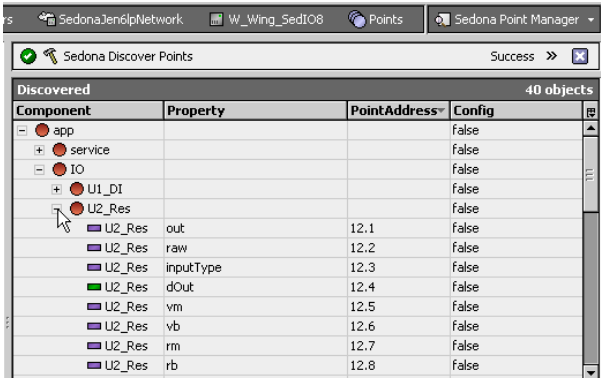


Table columns in the Discovered pane are as follows:

- **Component**
Name of the parent Sedona component or folder, from 1 to 7 characters maximum.
- **Property**
Sedona slot name for a property or action—i.e. how it is listed on the component’s slot sheet.
- **PointAddress**
Unique numerical ID of each Sedona component and child slot in the app, in form *compId.slotId*, for example 9.4 for the fourth slot of the ninth component.
- **Config**
Whether a slot’s value is persisted in an app save (true), or else not (false).

As in other point managers, to initiate an add you can double-click or drag-and-drop an item in the lower Database pane (for single items), or hold the Ctrl key down and click (select) multiple items to add with the **Add** button. This produces the **Add** dialog. See “Add dialogs in Sedona Point Manager”.

Add dialogs in Sedona Point Manager

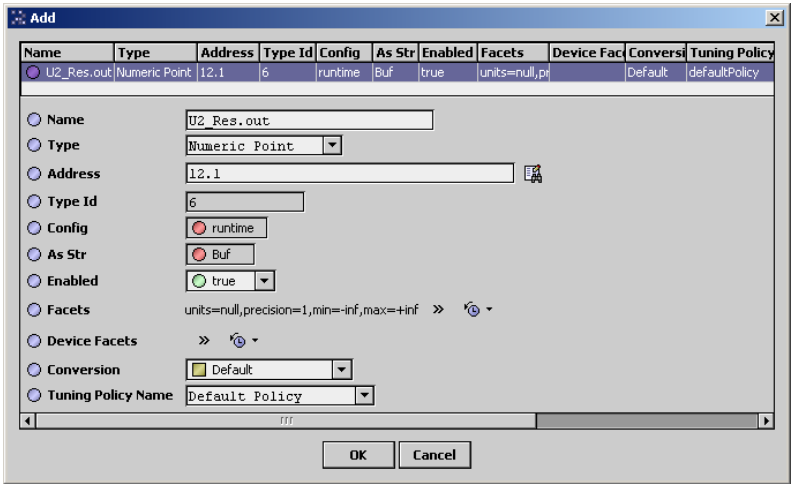
Depending if adding property items or action items, the **Add** dialog in the Sedona Point Manager varies.

- “Sedona proxy point Add dialog”
- “Sedona action point Add dialog”

Sedona proxy point Add dialog

An example **Add** dialog for a Sedona proxy point is shown in Figure 3-36 below.

Figure 3-36 Proxy point Add dialog in Sedona Point Manager



Data fields in the **Add** (and **Edit**) dialog for a proxy point contain most properties in the SedonaProxyExt, along with a few others. The fields are as follows:

- **Name**
Niagara name of the Sedona proxy point, which defaults to Sedona *ComponentName.slotName*, which can be left at default or changed if needed.

- **Type**
Niagara control point type for the proxy point, for example, Numeric Point, Numeric Writable, Boolean Point, Boolean Writable, and so on. Not editable after adding the proxy point.
Note: If the default, pre-selected type is a writable point type, you can safely choose a read-only point type instead. However, if the default type is a read-only point type (BooleanPoint, EnumPoint, etc.), it is not recommended to reselect a writable point type. Also see the property field “Config”.
- **Address**
Unique numerical ID of the property in the app, in form `compId.slotId`, for example 9.4 for the fourth slot of the ninth component.
Note: Automatically determined at proxy point add time. Manual editing is not recommended.
- **Type Id**
Read-only integer value, used internally to properly encode write values.
- **Config**
Either **runtime** or **config**.
 - If **runtime**, the property’s value is *not saved* to non-volatile Flash memory on an app save.
 - If **config**, an app save preserves the property’s value to non-volatile Flash memory.Often, “runtime” properties also have a “readonly” flag set—if so, typically the default proxy point type selected in the **Add** dialog a read-only type (BooleanPoint, NumericPoint, etc.).
- **AsStr**
Read-only value, either **Buf** or **AsStr**. If Buf, the slot’s value may be interpreted as a byte array or other data structure. If AsStr, the slot’s value may be interpreted as a null-terminated string. Applicable only if the Sedona property is type `sys::Buf`.
- **Enabled**
Either **true** (default) or **false**, to enable/disable the proxy point. If set to false, the point’s status is disabled, and any value polling is stopped.
- **Facets**
Point facets for the proxy point value, which are typically edited from defaults.
- **Device Facets**
Device facets for the source slot’s value, if any.
- **Conversion**
Specifies the conversion used between the “read value” (in Device Facets) and the parent point’s output (in selected point facets). Typically left at the “Default” selection.
- **Tuning Policy Name**
Specifies which of the network’s (SedonaNetwork or SedonaJen6lpNetwork) tuning policies should be used for reads and (if applicable) writes to this proxy point.
For related details see:
 - “SedonaNetwork tuning policy notes” on page 3-6
 - “SedonaJen6lpNetwork tuning policy notes” on page 3-14

Once added to the station, you can edit any of these fields if needed—with the exception of **Type**.

Sedona action point Add dialog

An example **Add** dialog for a Sedona action point is shown in [Figure 3-36](#) below.

Figure 3-37 Action point Add dialog in Sedona Point Manager

| Name | Type | Address | Type Id | Config | As Str | Enabled | Facets | Device Facet |
|------------------|---------------------|---------|---------|--------|--------|---------|--------|--------------|
| ● Ctr.resetCount | Sedona Action Point | 14.4 | | | | | | |

☒ Name: Ctr.resetCount
☐ Type: Sedona Action Point
☐ Address: 14.4
☐ Type Id: Cannot edit
☐ Config: Cannot edit
☐ As Str: Cannot edit
☐ Enabled: Cannot edit
☐ Facets: Cannot edit
☐ Device Facets: Cannot edit
☐ Conversion: Cannot edit
☐ Tuning Policy Name: Cannot edit

OK Cancel

There are only a few fields in the **Add** (and **Edit**) dialog for an action point, described as follows:

- **Name**
Niagara name of the Sedona action point, which defaults to `Sedona ComponentName . slotName`. This can be left at default or changed if needed.
- **Type**
Always **Sedona Action Point**, and not editable after adding point.
- **Address**
Unique numerical ID of the action in the app, in form `compId . slotId`, for example 14 . 4 for the fourth slot of the 14th component.
Note: Automatically determined at action point add time. Manual editing is not recommended.

For more details on action points, see [“About Sedona action points”](#) on page 3-41.

Sedona Point Manager "Database" notes

The “Database” table lists existing Sedona proxy points and action points, where each appears as a row in the table. Each proxy point represents one property of a Sedona component in that device; each action point represents a single action on a Sedona component ([Figure 3-38](#)).

Figure 3-38 Database shows Sedona proxy points and Sedona action points

| Name | Type | Out | Address | Type Id | Config | Fault Cause | Tuning Policy |
|-------------------|---------------------|-------------------|---------|---------|---------|-------------|---------------|
| FanStus.out | Boolean Point | Off {ok} | 11.4 | 128 | runtime | | defaultPolicy |
| Tempera.out | Numeric Point | 88.5 °F {ok} | 35.1 | 256 | runtime | | defaultPolicy |
| RLY3.out | Boolean Writable | true {ok} @ def | 27.1 | 128 | config | | defaultPolicy |
| app.save | Sedona Action Point | | 0.1 | | | | |
| Ctr.count | Numeric Writable | 2542.0 {ok} @ def | 14.1 | 256 | runtime | | defaultPolicy |
| RLY2.out | Enum Writable | On {ok} @ def | 26.1 | 128 | config | | defaultPolicy |
| Ctr.resetCount | Sedona Action Point | | 14.4 | | | | |
| plat.memAvailable | Numeric Point | 868.0 {ok} | 2.5 | 192 | runtime | | defaultPolicy |

You can resort items by clicking on any column header. This may be useful to sort by type, or perhaps by name.

Also, if you created point folders under a device's Points container, you can see all proxy points and action points in the device from the main (root) Points Manager using “All Descendants” tool. For details, see the section “All Descendants” in the *Drivers Guide*.

The following sections provide more details about the Database pane in the Sedona Point Manager:

- [Database point table columns](#)
- [Modifying the Database table](#)

Database point table columns

Note: Most data columns apply to proxy points, where only columns Name, Type, and Address are also applicable to Sedona action points.

By default, the following columns appear in the **Database** table:

- **Name**
Niagara name of the Sedona proxy point or action point. By default, names use the format: `ComponentName . slotName`. Names can be edited or left at default. If displayName are assigned to proxy points and action points (from the slot sheet of **Points**), they display here instead.
- **Type**
Niagara control point type for proxy points (**Boolean Point**, **Numeric Writable**, and so on) or if an action point, **Sedona Action Point**.
- **Out**
The current read property value, reflecting point facets, along with point status.
- **Address**
Unique numerical ID of the slot in the device's Sedona Framework app, in form `compId . slotId`, for example 9 . 4 for the fourth slot of the ninth component.
- **Type Id**
Read-only integer value, used internally to properly encode write values to the property.

- **Config**
Whether the target Sedona property value is saved to the device's non-volatile Flash memory upon a save to its Sedona Framework app. If the value is saved, "config" is indicated; otherwise Config is "runtime".
- **Fault Cause**
If a proxy point has a fault status, the fault cause text string explains why.
- **Tuning Policy**
The assigned tuning policy for the proxy point, by name of the network's tuning policy.

Note: You can also modify columns shown, see ["Modifying the Database table"](#) on page 3-38.

Modifying the Database table

You can modify which data columns appear in the Sedona Point Manager database table. For the options menu, simply click the small "table options" control in the upper right corner. See the section "Manager table features" in the *Drivers Guide* for general details.

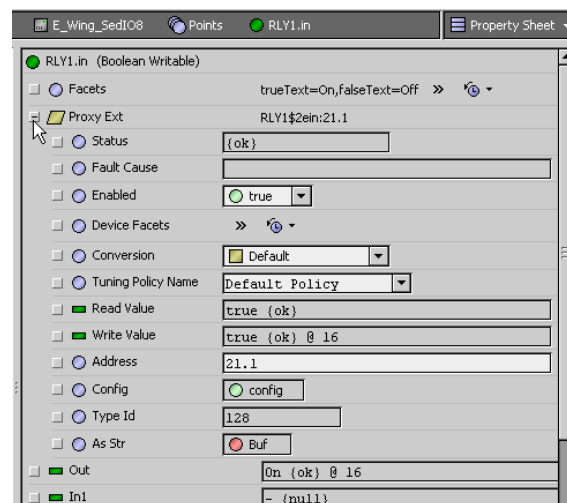
Non-default selections for data columns in the Sedona Point Manager Database table include various properties of both the parent proxy point and the SedonaProxyExt, and are the following:

- Path — Station path to the proxy point.
- AsStr — Whether target property is considered "Buf" or "AsStr".
- Enabled — Whether the Niagara proxy point is currently enabled for communications.
- Facets — Facets in use by the parent proxy point.
- Device Facets — Learned facets of source Sedona property, if any.
- Conversion — Conversion used between device facets and point facets (typically "Default").
- Read Value — Last value read from device, expressed in device facets.
- Write Value — (Applies to writable types only) Last value written, using device facets.

About Sedona proxy points

Sedona proxy points are similar to other driver's proxy points, meaning they are based on the same core Niagara control points, with the SedonaProxyExt. The SedonaProxyExt uses the same core properties of most driver's proxy points, such as Status, Fault Cause, Enabled, and so on. For general information, see the section "ProxyExt properties" in the *Drivers Guide*.

Figure 3-39 Example SedonaProxyExt in property sheet



The following SedonaProxyExt properties are of particular importance:

- **Tuning Policy Name**
Specifies which of the network's (SedonaNetwork or SedonaJen6lpNetwork) tuning policies should be used for reads and (if applicable) writes to this proxy point.
For related details see:
 - ["SedonaNetwork tuning policy notes"](#) on page 3-6
 - ["SedonaJen6lpNetwork tuning policy notes"](#) on page 3-14
- **Read Value**
(read only) Last value read from the device, expressed in device facets.

- **Write Value**
(read only) Applies if writable point only. Last value written, using device facets.
- **Address**
Unique numerical ID of the property in the app, in form `compId.slotId`, for example 9.4 for the fourth slot of the ninth component.
Note: Automatically determined at proxy point add time. Manual editing is not recommended.

For details on other SedonaProxyExt properties, see “[Sedona proxy point Add dialog](#)” on page 3-35.

Note: Sedona proxy points are modeled using standard Niagara data types. For details, see “[Sedona property data types to default Sedona proxy point types](#)” on page 3-39.

For a procedure on adding points, see “[Create Sedona proxy points \(and action points\)](#)” on page 2-10.

Sedona and Niagara data types and null notes

When comparing data types in Sedona and Niagara, note that properties in Sedona components use one of several primitive data types, including “bool” (boolean value), “int” (signed 32-bit integer), “float” (32-bit floating point), and several others. For complete details, see the documentation on the Sedona Framework org website, for example at <http://www.sedonadev.org/doc/primitives.html>.

Niagara models this data with Sedona proxy points, using standard control points based on “status value” data types (statusBoolean, statusNumeric, statusEnum, statusString). The following sections provide more details:

- [Sedona property data types to default Sedona proxy point types](#)
- [Niagara writable null status and Sedona](#)
- [Booleans as Enums](#)

Sedona property data types to default Sedona proxy point types

[Table 3-5](#) provides a cross listing of Sedona property data types, and the corresponding default Sedona (Niagara) proxy point types, and alternates.

Table 3-4 Sedona property data types to default Proxy Point types, with alternate types

| Sedona data type | Default Proxy Point types | Alternately available point types |
|---------------------------|-------------------------------|--|
| bool (boolean) | BooleanPoint, BooleanWritable | EnumPoint, EnumWritable (see “ Booleans as Enums ” on page 3-40) |
| float or double | NumericPoint, NumericWritable | — |
| int, long, byte, or short | EnumPoint, EnumWritable | NumericPoint, NumericWritable |
| Str (string), sys::Buf, | StringPoint, StringWritable | — |

Sedona does not explicitly model “multistate” (enumerated) data. However, note that properties using integer type primitives (int, long, byte, short) default to Enum point types. Often, say for properties using “int” or “long” data type, you may wish to change this to NumericPoint or NumericWritable at proxy point add time.

Niagara writable null status and Sedona

Writable Sedona proxy points (BooleanWritable, NumericWritable, EnumWritable, StringWritable) have a “Fallback” property that is, by default, set to “null”. When all other priority levels are cleared, a “null” status output results.

However, Niagara never issues any write to (or clears any value from) Sedona on a null transition—for example, whatever “null value” may exist in that Fallback property. Therefore, to remove any ambiguity you may want to clear (remove) the Fallback “null” setting, and enter (or verify) a specific fallback value to write from Niagara.

For example, consider a BooleanWritable proxy point for the “In” property of a Sedona “WriteBool” component. The BooleanWritable has a default Fallback property with the “null” status checked. If the proxy point is linked to a BooleanSchedule component that has its own “Default Output” of “- null”, it is possible that all inputs to the proxy point will be nulled. However, nothing will be written to Sedona on the transition from schedule “On” to “null”, such that the WriteBool will remain “true”.

Also note that Sedona has its own “null” implementation for the primitive bool (boolean) type. For this, there is no direct translation from a Niagara “null” to a Sedona “null”. However, if necessary the Sedona boolean null can be included in a Sedona proxy point. See “[Booleans as Enums](#)” for details.

Booleans as Enums

Although Sedona does not explicitly model “multistate” (enumerated) data, there is a unique exception of the boolean (bool) primitive type, with three possible states for boolean literals, as shown in Table 3-5.

Table 3-5 Sedona boolean (bool) states

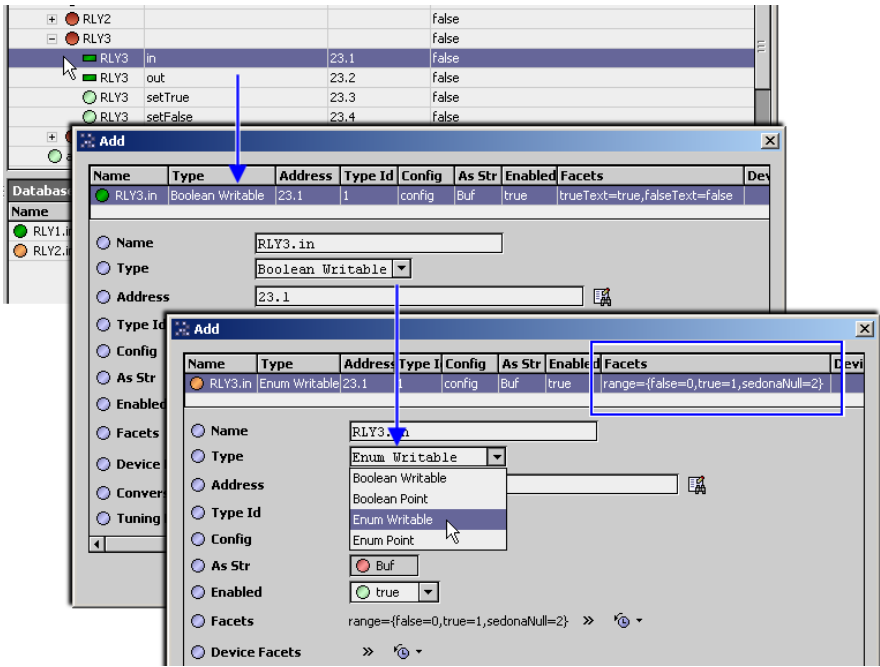
| Value | Ordinal | String |
|-------|---------|---------|
| false | 0 | “false” |
| true | 1 | “true” |
| null | 2 | “null” |

In Sedona, null can indicate an invalid boolean value. If used in a boolean expression, null evaluates to true (it is represented as 2 in memory). Not all Sedona Framework apps may make use of boolean null.

Note: The Niagara “null” status in writable Sedona proxy points does not translate into a “Sedona null”, nor does a Niagara null status ever result in any write (i.e. “null value”) to Sedona.

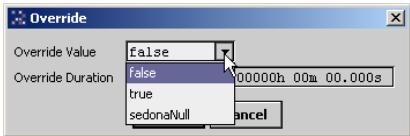
By default, a Sedona boolean property is proxied in Niagara as a BooleanPoint or (if writable) a BooleanWritable, with only two states. However, if you need to model a Sedona boolean property with all three states, you can select an EnumPoint (or if writable, an EnumWritable point) instead of a BooleanPoint or BooleanWritable, as shown being done in Figure 3-40.

Figure 3-40 Re-selecting an Enum point type instead of Boolean for a Sedona “bool” property



In this case, the Enum point’s facets are automatically configured using the value and ordinals shown in Table 3-5, except null appears as “sedonaNull” (so as to distinguish this from when point status is “null”). If needed, you can edit these descriptors in point facets.

Figure 3-41 Override action menu of EnumWritable that proxies a Sedona boolean (default descriptors)



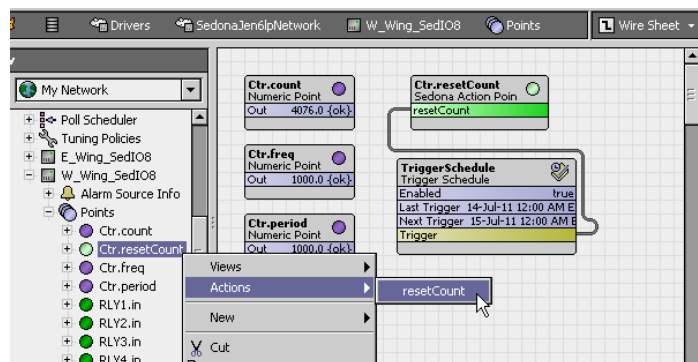
Note if an EnumWritable proxy point, these facet descriptors appear in the “override” action menu for the point, just like any other EnumWritable. See Figure 3-41.

About Sedona action points

Sedona action points are simple components, where each has a single action slot that represents a specific action on a Sedona component. You add and manage them from the Sedona Point Manager view of a Sedona Device's Points extension, along with Sedona proxy points.

In general, Sedona action point usage may be infrequent. One use case may be where a programmatic action is required from Niagara, such as the scheduled reset of an ongoing count (shown below).

Figure 3-42 Example Sedona action point linked into control logic



As shown in [Figure 3-42](#), the Sedona action is available as a right-click action in Niagara. It is also an available link target for a programmatic invoke from control logic—in this case from a TriggerSchedule.

CHAPTER 4

Sedona Framework Plugin Guides

Plugins (views) provide a visualization of components. There are many ways to view plugins. One way is directly in the tree. In addition, you can right-click on an item and select one of its views. Find documentation on a view by selecting **Help > On View** (F1) from the menu bar, or pressing F1 while in the view.

In this section, summary descriptions are given for views in the [jen6lp](#), [pansheet](#), and [sedonanet](#) modules, and a few details are given about views in the [nsedona](#) module. Summary descriptions typically include links to more detailed information.

Plugins in jen6lp module

The `jen6lp` module contains the following plugins (views):

- [Sedona Jen6lp Device Manager](#)

jen6lp-SedonaJen6lpDeviceManager

The **Sedona Jen6lp Device Manager** is the default view on a [SedonaJen6lpNetwork](#) component. It is used to manage [SedonaJen6lpDevice](#) children, where each represents a wireless Jennic-based device in the network where the JACE (with installed Sedona Jennic option card) acts as the “coordinator” node. This device manager offers online device discovery. For more details, see “[Sedona Jen6lp Device Manager view](#)” on page 3-20.

Plugins in nsedona module

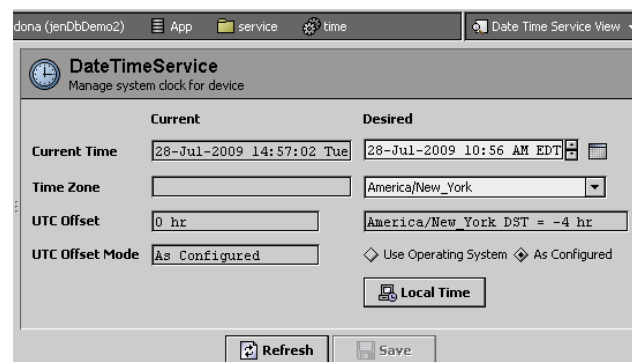
The `nsedona` module contains the following plugins (views), listed alphabetically:

- [Date Time Service View](#)
- [Sedona Link Sheet](#)
- [Sedona Property Sheet](#)
- [Sedona Slot Sheet](#)
- [Sedona User Manager](#)
- [Sedona Wire Sheet](#)

nsedona-DateTimeServiceView

The **Sedona Date Time Service** view the *default* view of the “time” service component (in the **service** folder), either a `DateTimeServiceJenRtc` or `DateTimeServiceStd` component.

Figure 4-1 Date Time Service View



This view provides a way to set the current time and date in the Sedona Framework node. Do this by typing and using the field controls for time and date (🕒 and 📅) and time zone (🌐), or by simply clicking the 🖱️ **Local Time** button to copy the Workbench PC's time and date to this node

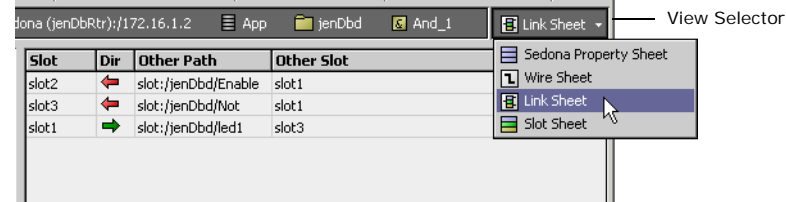
Note: After any **Save** in the Date Time Service View, also save the top-level “App” component.

nsedona-SedonaLinkSheet

🖱️ The **Sedona Link Sheet** view (link sheet) is an available view for any Sedona Framework component, providing details about any *link to* (inbound) or *link from* (outbound) that component.

Figure 4-2 shows an example link sheet for a component.

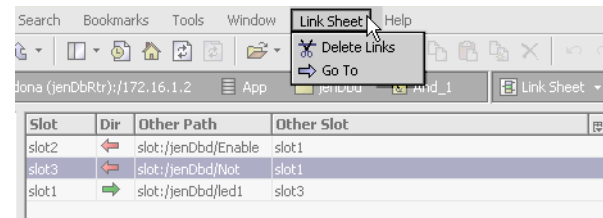
Figure 4-2 Example Sedona Link Sheet view (And component)



Each row represents a link, either an inbound link or an outbound link (seen in “Dir” column). Other columns provide information on the local slot (“Slot”), other component (“Other Path”), and the slot of that other component (“Other Slot”).

When a link sheet view is active, the Workbench menu bar includes a Link Sheet menu (Figure 4-3).

Figure 4-3 Link Sheet tool bar menu



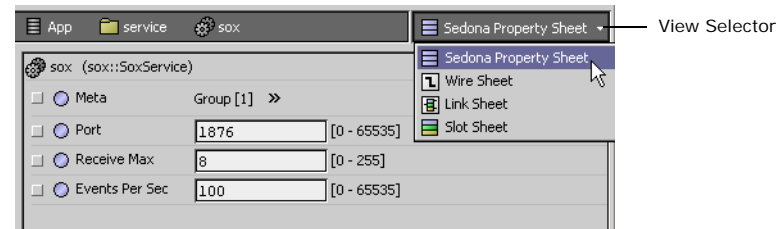
Tool bar menu items to **Delete Links** (one or more links selected) or **Go To** linked component (single link selected) are also available in the link sheet via a right-click menu.

nsedona-SedonaPropertySheet

🖱️ The **Sedona Property Sheet** view (property sheet) is a standard view for any Sedona Framework component, providing access to the *properties* for that component. If a container type component, it also includes any child components as expandable [+] nodes—for example, the property sheet for a Folder component, will include a node for each component that it contains.

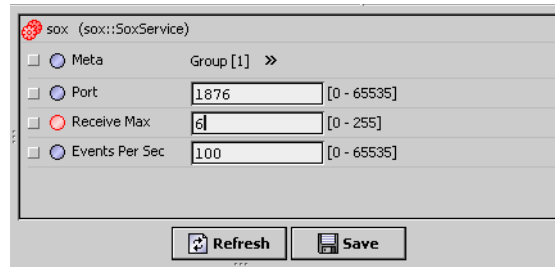
For many components, the property sheet view is the *default* view. The default view appears at the *top* of the “View Selector” for a component, as shown in Figure 4-4 (property sheet for SoxService).

Figure 4-4 Example Sedona Property Sheet view (SoxService)



In a property sheet, some properties may appear with a gray (vs. white) background—these are typically read-only properties—a common example is the property “Out”. When you click on a *writable* property and enter a value, its appearance changes red to indicate a save is needed, and the **Save** button at the bottom of the view becomes available (Figure 4-5).

Figure 4-5 Unsaved (ineffective) property sheet change indicated by red



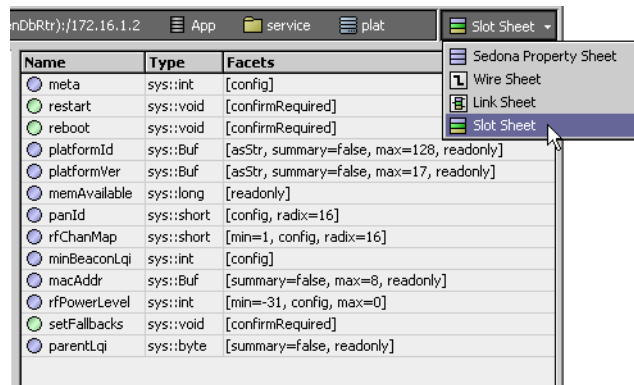
Click the **Save** button to write the change to the node's RAM, making it effective. However, note that such changes are not persisted in Flash memory of the device unless you save the App node.

For details on the “Meta” property in the Sedona property sheet of any component, see “[nsedona-Sedona-Component \(anyKit:anyComponent\)](#)” on page 5-3

nsedona-SedonaSlotSheet

The **Sedona Slot Sheet** view (slot sheet) is a standard view for any Sedona Framework component, providing read-only data about both *properties* and *actions* for that component. [Figure 4-6](#) shows an example slot sheet for a component.

Figure 4-6 Example Sedona Slot Sheet view (jennic:JennicRouter component)



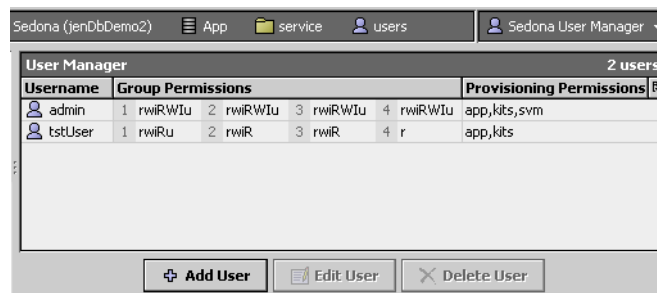
Names of slots use a camelCase format, for example “minBeaconLqi”. Whereas, those same slots appear listed on property sheet views by converting this format to include capital letters and spaces, for example “Min Beacon Lqi”.

The “Type” column shows a property's primitive data type (where applicable), and the “Facets” column shows other data, such as if an action produces a confirmation dialog “[confirmRequired]”.

nsedona-SedonaUserManager

The **Sedona User Manager** view is the *default* view of the “users” service component (in the **service** folder) of a Sedona Framework app (the sys:UserService component).

Figure 4-7 Sedona User Manger view



This view provides a way to view, add, edit, and delete child users in the app. Properties of each user determine their app login credentials, as well as their Sedona component permissions (across four groups), and sox provisioning permissions (across three different areas).

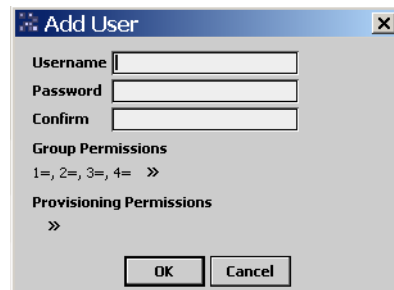
This view includes a [User table](#) and three [User Manager buttons](#).

User table The table area lists each user on a separate row, summarizing in columns:

- **Username**
Name of app user (7 characters maximum)
- **Group Permissions**
Available permissions for that user among component groups 1-4, where:
Operator level permissions show as “r” (read), “w” (write), “i” (invoke, action)
Admin level permissions show as “R” (Read), “W” (Write), “I” (Invoke, action), “u” (user)
- **Provisioning Permissions**
Available sox provisioning permissions for that user among three areas, where:
Sedona app provisioning is “app”, kit provisioning is “kits”, and Sedona VM provisioning is “svm”

User Manager buttons The  **Add User** button is always available. An Add User dialog results (Figure 4-8).

Figure 4-8 Add User dialog



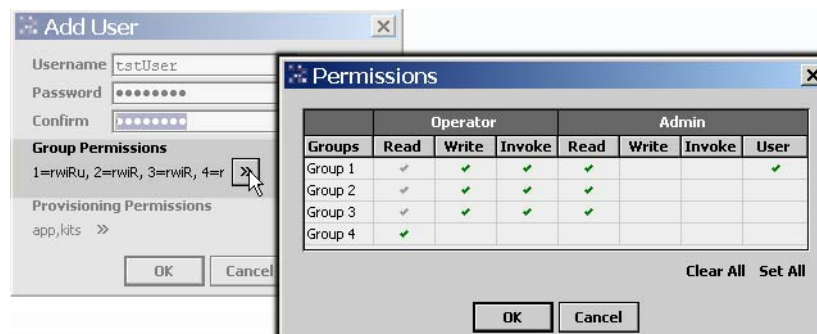
The Add User dialog box contains the following fields and buttons:

- Username**: Text input field.
- Password**: Password input field.
- Confirm**: Password input field.
- Group Permissions**: Section with a label "1=, 2=, 3=, 4=" and a right-pointing arrow button.
- Provisioning Permissions**: Section with a right-pointing arrow button.
- OK** and **Cancel** buttons at the bottom.

Configuration fields in Add User dialog are as follows:

- **Username**
Unique name of user, 7 alphanumeric characters maximum, and is case sensitive. Must begin with letter and have no spaces or punctuation, apart from underscore (_).
- **Password**
Login password for this user, case sensitive.
- **Confirm**
Repeat of the same password for this user, case sensitive.
- **Group Permissions**
Component permissions for the 4 different component groups can be enabled or disabled by clicking in the popup Permissions dialog, with two access levels: Operator and Admin (Figure 4-9).

Figure 4-9 Group Permissions dialog for a User



The Group Permissions dialog box shows the Add User dialog in the background with the Group Permissions section expanded. The Permissions popup dialog is overlaid on top, showing a table of permissions for four groups.

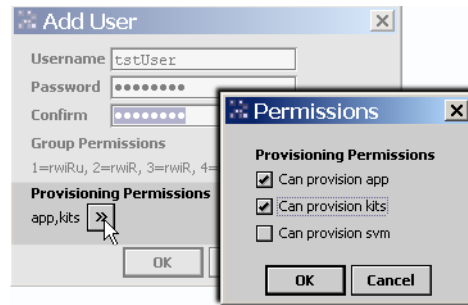
| Groups | Operator | | | Admin | | | User |
|---------|----------|-------|--------|-------|-------|--------|------|
| | Read | Write | Invoke | Read | Write | Invoke | |
| Group 1 | ✓ | ✓ | ✓ | ✓ | | | ✓ |
| Group 2 | ✓ | | ✓ | ✓ | | | |
| Group 3 | ✓ | ✓ | ✓ | ✓ | | | |
| Group 4 | ✓ | | | | | | |

Buttons: **Clear All**, **Set All**, **OK**, **Cancel**.

If a component belongs to more than one group, the user has the highest of those different group permissions for that component.

- **Provisioning Permissions**
Sox provisioning permissions for the 3 different areas of the platform can be enabled or disabled by clicking in the popup Permissions dialog (Figure 4-10).

Figure 4-10 Provisioning Permissions dialog for a User



The **Edit User** button is available when any one user (row) is selected. It produces an **Edit User** dialog that is similar to the **Add User** dialog (Figure 4-8), where changes can be made to any field *except* Username.

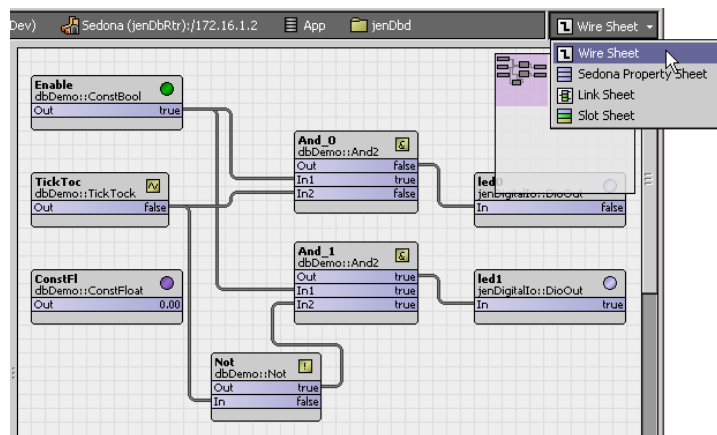
The **Delete User** button is available when any one user (row) is selected. It produces an **Delete User** confirmation dialog. If confirmed with **Yes**, that child User component is deleted.

nsedona-SedonaWireSheet

The **Sedona Wire Sheet** view (wire sheet) is a standard view for any Sedona Framework component. However, the main usage is with container-type components (notably Folder).

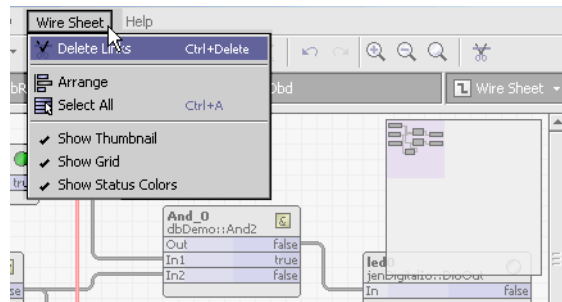
The wire sheet view is the *default* view for a Folder component, as shown in Figure 4-11.

Figure 4-11 Example Sedona Wire Sheet view (Folder)



Wire sheet views are used to move components and link together, to graphically define control logic. When a wire sheet view is active, the Workbench menu bar includes a Wire Sheet menu (Figure 4-12).

Figure 4-12 Wire Sheet tool bar menu



Tool bar menu items to **Delete Links** (one or more links selected), **Arrange** shapes, and **Select All** are also available in the wire sheet via a right-click menu. Enable/disable options for showing the navigation thumbnail, grid, and status colors are only in the tool bar menu.


Note: To change the default wire sheet view settings, including the thumbnail, from the Workbench **Tools** menu, click the **Options** entry, then click **Wire Sheet** in the left-side options tree.

Plugins in pansheet module

The `pansheet` module contains the following plugins (views):

- [Pan Sheet](#)

pansheet-PanSheetView


 The **Pan Sheet** is an available diagnostic view on a [SedonaJen6lpNetwork](#). It provides a graphical, hierarchical representation of the “JenNet” tree structure of the wireless network. The view also includes a “Pan Network” table showing “Pan Info” data collected throughout the network. Pan info data comes from [SedonaJen6lpNeighborEntry](#) slots in the `SedonaJen6lpNetwork` and all of its child `SedonaJen6lpDevices`. For details in this document, see “[Pan Sheet view of SedonaJen6lpNetwork](#)” on page 3-22. Also refer to the *Jennic Network Visualization (Pan Sheet) - Engineering Notes* document.

Plugins in sedonanet module


The `sedonanet` module contains the following plugins (views):

- [Chopan Device Manager](#)
- [Chopan Point Manager](#)
- [Manifest Manager](#)
- [Sedona Device Manager](#)
- [Sedona Point Manager](#)


sedonanet-ChopanDeviceManager

 The **Chopan Device Manager** is the default view of a **Virtual Chopan Network** under a `SedonaJen6lpDevice`'s **Chopan Virtual** gateway. In this view, you can discover and create “Chopan devices”, which result in *Sedona components* created in the remote Jennic-based device. Chopan devices (and Chopan points below them) act as the “Chopan client” interface in the device. For an overview in this document, see “[About the Chopan Virtual gateway](#)” on page 3-26. For complete details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-ChopanPointManager


 The **Chopan Point Manager** is the default view of a **Virtual Chopan Device** in a **Virtual Chopan Network** under a `SedonaJen6lpDevice`'s **Chopan Virtual** gateway. In this view, you can discover and create “Chopan points”, which result in *Sedona components* created in the remote Jennic-based device. Chopan points act as the “Chopan client” interface in the device. For an overview in this document, see “[About the Chopan Virtual gateway](#)” on page 3-26. For complete details, refer to the *Sedona Framework Chopan Usage* document, including the section “Chopan Point Manager notes”.

sedonanet-ManifestManager


 Use the **Manifest Manager** to manage Sedona kit *manifest* files, which are necessary for Sox access of Sedona Framework devices, both by a JACE station and by Workbench. The Manifest Manager is an available view on a Sedona Framework network component (`SedonaNetwork` or `SedonaJen6lpNetwork`) for managing manifests on a JACE. Sedona-enabled Workbench also has Sedona Manifest Manager (menu item **Tools > Sedona Manifest Manager**), for managing manifests on the Workbench host.

For a quick start procedure using the network-level **Manifest Manager**, see “[Run the Manifest Manager to ensure kit manifests are loaded](#)” on page 2-8. For complete Manifest Manager details, refer to the document *Sedona Framework Manifest Manager - Engineering Notes*.

sedonanet-SedonaDeviceManager

 The **Sedona Device Manager** is the default view on a [SedonaNetwork](#) component, as well as a `SedonaDeviceFolder`. It is used to manage [SedonaDevice](#) children, where each represents an Ethernet/IP equipped Sedona Framework device. This device manager does not offer online device discovery—instead you manually add devices using the **New** button and **New** dialogs. For more details, see “[Sedona Device Manager view](#)” on page 3-7.

sedonanet-SedonaPointManager

 The **Sedona Point Manager** is the default view on the `SedonaPointDevExt (Points)` extension of any Sedona device component (`SedonaDevice`, `SedonaJen6lpDevice`). In this view, you can discover, create, edit, and delete Sedona proxy points and action points. The Sedona Point Manager is also the default view for any `SedonaPointFolders` under the Points container. For more details, see “[About the Sedona Point Manager](#)” on page 3-33.

CHAPTER 5

Sedona Framework Component Guides

Summary information on Sedona Framework-related NiagaraAX components is provided in alphabetical order, for components found in the following modules:


- [jen6lp](#)
- [nsedona](#)
- [sedonanet](#)

Components in jen6lp module

The `jen6lp` module contains the following components, listed alphabetically:


- [Ipv4Mac154Table](#)
- [PanInfoPoller](#)
- [SedonaJen6lpDevice](#)
- [SedonaJen6lpDeviceFolder](#)
- [SedonaJen6lpNeighborEntry](#)
- [SedonaJen6lpNetwork](#)
- [SedonaJen6lpPingMonitor](#)

jen6lp-Ipv4Mac154Table

 `Ipv4Mac154Table` (**Address Map**) is a child container under a `SedonaJen6lpNetwork`. Its configuration property “Ipv4 Address Base” defines the private IPv4 subnet to which Jennic-based nodes are remapped by the JACE coordinator. It also contains dynamically-created “entryN” container slots that define each node’s mapping, including specific IPv4 address, MAC 15.4 address, and JenNet “join” status. An available “Clear Table” action clears out all “entryN” children—used only if reconfiguring the network.

For more details, see “[SedonaJen6lpNetwork coordinator properties](#)” on page 3-16.


jen6lp-PanInfoPoller

 The `PanInfoPoller` (**PanInfo Poller**) is a frozen container slot of a `SedonaJen6lpNetwork`. It specifies how “PAN info” (JenNet related) data is received, either by ongoing polling or “on demand”. The default is polling disabled (property `Enabled` = false), with property “Use Chopan” as true (vs. Sox). Chopan usage requires that Jennic-based devices have a Sedona Framework app with both the `PanInfoService` and the `ChopanService` installed, as well as the `PanInfoServlet` added under the `ChopanService`.


PAN info from child nodes is modeled dynamically under the `SedonaJen6lpNetwork`’s frozen container component **Child Pan Info**, in `SedonaJen6lpNeighborEntry` components. The available Pan Sheet view of the `SedonaJen6lpNetwork` provides a graphical representation of such paninfo.

For more details, see “[SedonaJen6lpNetwork Pan Info properties](#)” on page 3-18, “[Pan Sheet view of SedonaJen6lpNetwork](#)” on page 3-22, and “[SedonaJen6lpDevice properties](#)” on page 3-23.

jen6lp-SedonaJen6lpDevice

 `SedonaJen6lpDevice` represents a wireless Jennic-based device, and is the only valid type of device object in a `SedonaJen6lpNetwork`. As a subclass of the `SedonaDevice`, it has the same common device properties and Points extension, plus additional properties and container slots for support of wireless Jennic communications as well as Chopan. For more details, see “[About the SedonaJen6lpDevice component](#)” on page 3-23.

jen6lp-SedonaJen6lpDeviceFolder

 `SedonaJen6lpDeviceFolder` is the implementation of a device folder under a `SedonaJen6lpNetwork`. You can use these folders to organize `SedonaJen6lpDevice` components in the network.

Typically, you add such folders using the **New Folder** button in the Sedona Jen6lp Device Manager view of the SedonaJen6lpNetwork. Each SedonaJen6lpDeviceFolder has its own Sedona Jen6lp Device Manager view. The SedonaJen6lpDeviceFolder is also available in the `jen6lp` palette.

jen6lp-SedonaJen6lpNeighborEntry


- A SedonaJen6lpNeighborEntry (MAC802.15.4MacAddress) is a dynamic child container that stores “paninfo” data about a nearby (neighbor) Jennic node. Its read-only properties report if the device is a sleeping end device, its MAC 15.4 address, and several numerical statistics including a link quality index, and numbers of packets lost, sent, and received. These components can be either “child” or “parent” in nature, depending on the context of component hierarchy.
- In the SedonaJen6lpNetwork, its frozen **Child Pan Info** slot holds a number of child SedonaJen6lpNeighborEntries, one for each direct child node. Because the network represents the (top level) coordinator, it has no parent entries.
- Each SedonaJen6lpDevice has a **Parent Pan Info** slot, of type SedonaJen6lpNeighborEntry, with data about its one (and only) Jennic parent node. Depending on the device’s location and the network’s “depth”, this parent node may either be the coordinator (JACE option card), or another Jennic-based device acting as a “router” node.
Each SedonaJen6lpDevice also has a **Child Pan Info** container slot, which may (or may not) have SedonaJen6lpNeighborEntry children.

The entire collection of data from all SedonaJen6lpNeighborEntries, network-wide, is reflected *graphically* in the Pan Sheet view of the SedonaJen6lpNetwork. This can be useful for diagnostic purposes. For related details in this document, see the following sections:

- “SedonaJen6lpNetwork Pan Info properties” on page 3-18.
- “SedonaJen6lpNetwork actions” on page 3-19.
- “Pan Sheet view of SedonaJen6lpNetwork” on page 3-22.
- “SedonaJen6lpDevice properties” on page 3-23.
- “SedonaJen6lpDevice actions” on page 3-25.

Find complete details about using the **Pan Sheet** in the *Jennic Network Visualization (Pan Sheet) - Engineering Notes* document.

jen6lp-SedonaJen6lpNetwork


 SedonaJen6lpNetwork is a subclass of the base SedonaNetwork, used for managing a network of wireless (802.15.4) Jennic-based devices in a station running in a JACE with an installed Sedona Jennic option card. Devices are represented by child SedonaJen6lpDevice components, mapped from their native IPv6 (6LoWPAN) addresses to a private IPv4 address range defined in the network.

The SedonaJen6lpNetwork extends the base SedonaNetwork, providing additional properties, child containers, and views that help configure and troubleshoot the Jennic wireless network.

As with other NiagaraAX driver networks, the SedonaJen6lpNetwork should reside under the station’s Drivers container. The default view of the network is the **Sedona Jen6lp Device Manager**.

For more details, see “About the SedonaJen6lpNetwork component” on page 3-11 and “Sedona Jen6lp Device Manager view” on page 3-20.

jen6lp-SedonaJen6lpPingMonitor


 The SedonaJen6lpPingMonitor (**Monitor**) is a frozen container slot of a SedonaJen6lpNetwork. It periodically checks the Jennic node status of devices in the local coordinator to determine the corresponding child Jen6lpDevices’ health. PingMonitor provides built-in support to generate alarms when these pingables are down. See “About Monitor” in the *Drivers Guide* for general monitor details.

Components in nsedona module

The nsedona module contains the following components, listed alphabetically:

- **DaspTunnel** (SoxTunnel)
- **SedonaComponent** (*anyKit:anyComponent*)

nsedona-DaspTunnel

 DaspTunnel (**SoxTunnel**) is the station service that allows Sox tunneling through the running (JACE) station to a Sedona Framework device. This allows Niagara Workbench to access Sox Tools and app configuration for Sedona Framework devices that are modeled in the station, in either (or both) a SedonaNetwork or SedonaJen6lpNetwork. The best practice is to copy the SoxTunnel into the station’s Services folder.

Note: To use this feature, the NiagaraAX station host as well as any Niagara Workbench client must have a license with the tunneling feature, with its attribute `sox="true"`.

The SoxTunnel service contains several properties, often which can be left at defaults. For complete details, refer to the document *Sedona Framework Sox Tunneling - Engineering Notes*.

nsedona-SedonaComponent (*anyKit:anyComponent*)

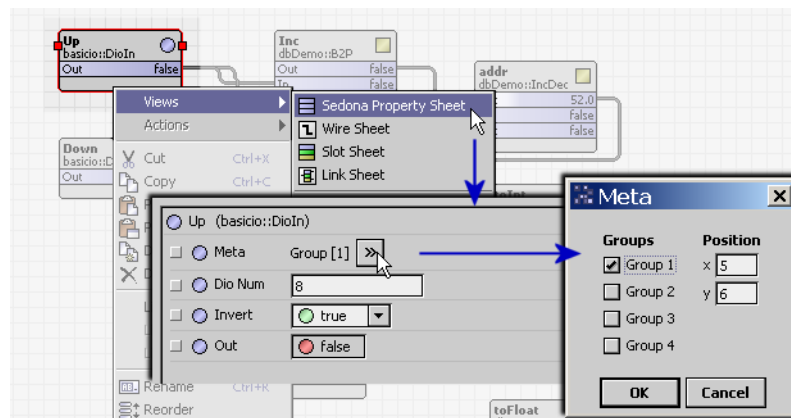
Represents any Sedona Framework component. All Sedona Framework components provide the identical “Guide on Target” Help ord in Workbench, instead of unique ords by *kitName-componentName*. Therefore this section describes the single common property implemented by all Sedona Framework components:**Meta**.

Note: Refer to the vendor’s documentation for any Sedona Framework device for specific details on Sedona components used in custom kits, and also the Sedona Framework Developer’s website for details on components in open source kits. At the time of this document, the URL for this API documentation is:

<http://www.sedonadev.org/doc/api.html>

Meta The Meta property is available in property sheet of any Sedona Framework component, with Figure 5-1 showing an example.

Figure 5-1 Meta property example in Sedona component



Meta stores two different pieces of information:

- **Groups**
Four component groups are available, numbered 1 through 4. Each component *must* belong to at least one group, and can belong to all four if desired. Membership in a group determines what permissions an app *user* has for that component, according to that user’s permissions map. For related details, see “[nsedona-SedonaUserManager](#)” on page 4-3.
- **Position**
The x and y coordinates of the component’s upper-left corner’s glyph (shape) in the wire sheet view of its parent container. Integer values are used, which correspond to grid increments on the wire sheet, starting with 0, 0 at the wire sheet’s top-left corner.
A component’s position values automatically track any “drag” movement of its shape on a wire sheet, and thus typically are not directly edited in the Meta dialog (Figure 5-1).
Storing the position of the components in an app maintains wire sheet layouts, which is essential when working with collections of components and the links between them.

Components in sedonanet module

The `sedonanet` module contains the following components, listed alphabetically:

- [ChopanServer](#)
- [ChopanVirtualGateway](#)
- [SedonaActionPoint](#)
- [SedonaDevice](#)
- [SedonaDeviceFolder](#)
- [SedonaNetwork](#)
- [SedonaPollScheduler](#)
- [SedonaPointDeviceExt](#)
- [SedonaPointFolder](#)
- [SedonaTuningPolicy](#)
- [SedonaTuningPolicyMap](#)
- [VirtualChopanBooleanPoint](#)
- [VirtualChopanDevice](#)
- [VirtualChopanFloatPoint](#)
- [VirtualChopanNetwork](#)

sedonanet-ChopanServer

- ChopanServer is a container slot under a SedonaJen6lpNetwork for configuration of the JACE station's CHoPAN server capability. Its properties specify the UDP port monitored, and whether the server is enabled or disabled. Debug output to the station's Application Director can also be enabled.

If configured for Chopan in their Sedona Framework app, this allows Jennic-based devices in the SedonaJen6lpNetwork to access station data via Chopan client requests. For more details, see [“SedonaJen6lpNetwork Chopan Server”](#) on page 3-16. For complete Chopan details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-ChopanVirtualGateway

- 🔌 The ChopanVirtualGateway is the SedonaJen6lpNetwork implementation of the Baja Virtual Gateway. A virtual gateway is a component that resides under the station's component space (`Config`), and acts as a gateway to the station's “virtual component space.”

Each SedonaJen6lpDevice has its own ChopanVirtualGateway, at the same level as its device extensions (Points, in this case). Using the gateway, you can discover nodes in the network that have CHoPAN servers in them - including the JACE. You can add Sedona components to the node's app that represent these CHoPAN servers, and then add components inside the servers that represent points within them. These are actual components inside the app of the physical device your SedonaDevice is mapping. However, they are presented in the form of Niagara virtual points, because the device and point addition is not taking place in the station, but on the device itself.

For details in this document, see [“About the Chopan Virtual gateway”](#) on page 3-26. For complete Chopan details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-SedonaActionPoint

- SedonaActionPoint is Niagara representation for a particular action slot of a Sedona component, as added under the **Points** extension of a SedonaDevice or SedonaJen6lpDevice using its **Sedona Point Manager** view. The SedonaActionPoint provides a single right-click action, which proxies the action available on the component in the device's Sedona Framework app.

Both Sedona proxy points and Sedona action points are visible in the Sedona Point Manager view of a Sedona Framework device. For more details, see [“About Sedona action points”](#) on page 3-41.

sedonanet-SedonaDevice


- 🖨 SedonaDevice represents an Ethernet/IP (or 802.11 WiFi) capable device, and is the only valid device object type in a SedonaNetwork. As the base class Sedona Framework device component, it has common device properties and a Points extension. See [“About the SedonaDevice component”](#) on page 3-10.

sedonanet-SedonaDeviceFolder

- 📁 SedonaDeviceFolder is the implementation of a device folder under a SedonaNetwork. You can use these folders to organize SedonaDevice components in the network.

Typically, you add such folders using the **New Folder** button in the Sedona Device Manager view of the SedonaNetwork. Each SedonaDeviceFolder has its own Sedona Device Manager view. The SedonaDeviceFolder is also available in the `sedonanet` palette.


sedonanet-SedonaNetwork

 SedonaNetwork is the base class Sedona Framework network, used for managing a network of Sedona Framework devices equipped for Ethernet/IP. If properly licensed, the NiagaraAX station can run in a JACE or a Supervisor station. Devices are represented by child SedonaDevice components, the only valid device type for this network type.


As with other NiagaraAX driver networks, the SedonaNetwork should reside under the station's Drivers container. The default view of the SedonaNetwork is the **Sedona Device Manager**, where you manually add devices, specifying each device's unique IPv4 address.

For more details, see [“About the SedonaNetwork component”](#) on page 3-5 and [“Sedona Device Manager view”](#) on page 3-7.


sedonanet-SedonaPollScheduler

 The SedonaPollScheduler is the Sedona implementation of a poll scheduler in a SedonaNetwork (or SedonaJen6lpNetwork). Operation is similar to most other polling networks.


sedonanet-SedonaPointDeviceExt

 SedonaPointDeviceExt is the Sedona implementation of PointDeviceExt. Its primary view is the **Sedona Point Manager**. Currently it is the only device extension for any Sedona Framework device (SedonaDevice, SedonaJen6lpDevice). It serves as the container for all Sedona proxy points for a specific device. For more details, see [“About the Sedona Point Manager”](#) on page 3-33.


sedonanet-SedonaPointFolder

 SedonaPointFolder is the Sedona implementation of a folder under a Sedona Framework device's **Points** container (SedonaPointDeviceExt). You add such folders using the **New Folder** button in the **Sedona Point Manager** view of the Points component. Each SedonaPointFolder has its own Sedona Point Manager view. The SedonaPointFolder is also available in the sedonanet palette.


sedonanet-SedonaProxyExt

 SedonaProxyExt is the proxy extension for any type of Sedona proxy point, as added under the **Points** extension of a SedonaDevice or SedonaJen6lpDevice using its **Sedona Point Manager** view. The SedonaProxyExt contains common ProxyExt properties, as well as a few unique to Sedona proxy points. For more details, see [“About Sedona proxy points”](#) on page 3-38.


sedonanet-SedonaTuningPolicy

 A tuning policy for a SedonaNetwork (or SedonaJen6lpNetwork), with standard NiagaraAX tuning policy properties. For more details, see [“SedonaNetwork tuning policy notes”](#) on page 3-6 and [“SedonaJen6lpNetwork tuning policy notes”](#) on page 3-14.


sedonanet-SedonaTuningPolicyMap

 SedonaTuningPolicyMap (**Tuning Policies**) is a SedonaNetwork (or SedonaJen6lpNetwork) container slot for one or more [SedonaTuningPolicy](#)(ies).


sedonanet-VirtualChopanDevice

 VirtualChopanDevice represents a “ChopanDevice” component in the Sedona Framework app of a remote Jennic-based device. In the JACE station, it is a child of the VirtualChopanNetwork under the ChopanVirtualGateway of a SedonaJen6lpDevice. It can contain child virtual “Chopan points” (Boolean or float types), where each represents a data item served in Chopan from another Jennic-based device (or the JACE station). The default view of a VirtualChopanDevice is the Chopan Point Manager, where you can discover and add virtual Chopan points. For an overview, see [“About the Chopan Virtual gateway”](#) on page 3-26. For more details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-VirtualChopanBooleanPoint


 VirtualChopanBooleanPoint is one of two types of “Chopan points” (the other type is VirtualChopanFloatPoint). It represents a Boolean type (Chopan client) component in the Sedona Framework app of a remote Jennic-based device. In the JACE station, it is a child of a VirtualChopanDevice in a ChopanVirtualNetwork within the ChopanVirtualGateway of a SedonaJen6lpDevice. The source (Chopan server) device is another Jennic-based device, or if enabled, the JACE station via the Chopan server slot of the SedonaJen6lpNetwork. For an overview, see [“About the Chopan Virtual gateway”](#) on page 3-26. For more details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-VirtualChopanFloatPoint

 VirtualChopanFloatPoint is one of two types of “Chopan points” (the other type is VirtualChopanBooleanPoint). It represents a float type (Chopan client) component in the Sedona Framework app of a remote Jennic-based device. In the JACE station, it is a child of a VirtualChopanDevice in a Chopan-

VirtualNetwork within the ChopanVirtualGateway of a SedonaJen6lpDevice. The source (Chopan server) device is another Jennic-based device, or if enabled, the JACE station (via the Chopan server slot of the SedonaJen6lpNetwork). For an overview, see [“About the Chopan Virtual gateway”](#) on page 3-26. For more details, refer to the *Sedona Framework Chopan Usage* document.

sedonanet-VirtualChopanNetwork

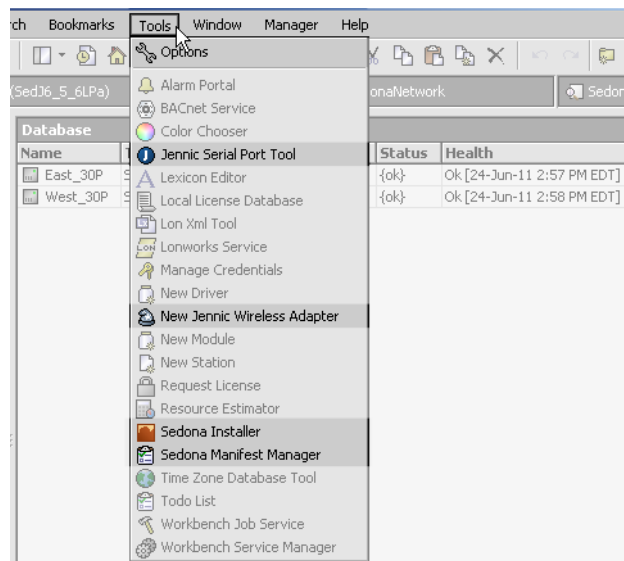
 VirtualChopanNetwork is a “virtual component” child of the ChopanVirtualGateway in a SedonaJen6lpDevice. It represents the “ChopanNetwork” component in the Sedona Framework app of a remote Jennic-based device. In the JACE station, it can contain child VirtualChopanDevice components, each representing a Jennic-based device with Chopan installed, as well as one device representing the JACE. The default view of the VirtualChopanNetwork is the Chopan Device Manager, where you can discover and add these VirtualChopanDevice components. For an overview, see [“About the Chopan Virtual gateway”](#) on page 3-26. For more details, refer to the *Sedona Framework Chopan Usage* document.

APPENDIX A

Sedona Framework Tools for Workbench

If Workbench has been enabled for Sedona Framework TXS 1.1 (using the Workbench tool **Sedona Installer**), several Sedona Framework-related “tools” become available in the Workbench **Tools** menu. This section provides overview information on these tools, along with links to any other documents with more details.

Figure A-1 Sedona Framework-related tools in Tools menu of Workbench



The following Sedona Framework-related Workbench tools are included:

- [“Jennic Serial Port Tool”](#) on page A-2
- [“New Jennic Wireless Adapter”](#) on page A-2
- [“Sedona Installer”](#) on page A-3
- [“Sedona Manifest Manager”](#) on page A-3

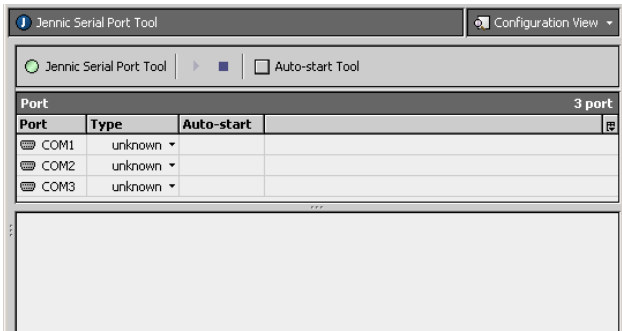
Note: The *Sedona Installer* is already included in any Workbench distribution, and is how you first enable Niagara Workbench for Sedona Framework TXS 1.1. Afterwards, you also use it to “upgrade” the Sedona Framework-specific files and NiagaraAX modules for your Workbench.

Jennic Serial Port Tool

The **Jennic Serial Port Tool** allows you to reconfigure a wireless Sedona Jennic USB coordinator (USB stick), if available, for direct Sox connection from Workbench to a single Jennic-based device. For example, to change its Panid, channel, or debug level.

This tool also provides a “flasher” utility to download firmware into a serially attached Jennic-based device (special cable typically required), or to update the firmware in a Sedona Jennic USB coordinator.

Figure A-2 Jennic Serial Port Tool

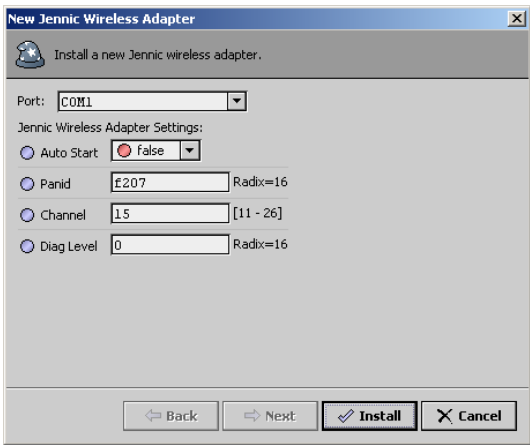


For more details, refer to the document *Jennic Serial Tools Guide*.

New Jennic Wireless Adapter

The **New Jennic Wireless Adapter** tool is a wizard you can use to identify and configure a new wireless Sedona Jennic USB coordinator (USB stick) for use with your Sedona Framework TXS enabled Niagara Workbench PC.

Figure A-3 New Jennic Wireless Adapter wizard



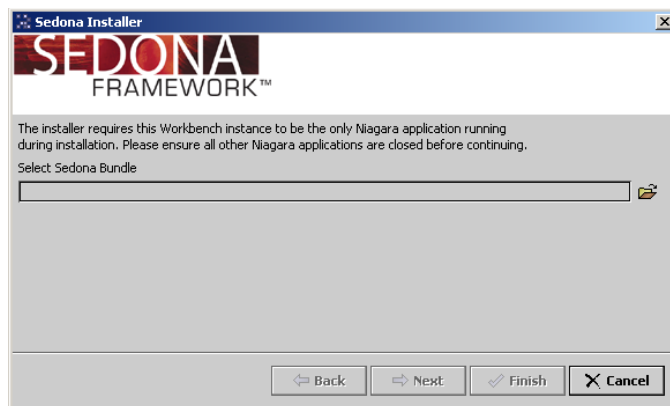
Note: Prior to using this tool, you need to download and install the latest FTDI (Future Technologies Devices International) “VCP driver” for your system. After you use the Windows Device Manager to determine the COM port number assigned to this “virtual com port”, use this tool to configure the USB coordinator. For more details, refer to the document *Jennic Serial Tools Guide*.

Sedona Installer

The **Sedona Installer** is the tool you use to upgrade a standard installation of Niagara Workbench to be “Sedona-enabled” Niagara Workbench, using a “Sedona bundle” image zip file. After this initial installation, you continue to use the Sedona Installer for applying Sedona bundle updates to your Workbench platform.

Note: *Sedona includes licensed features for Workbench as well as JACE hosts with stations that use Sedona Framework network drivers.*

Figure A-4 Sedona Installer is used to install “Sedona bundle” image distributions

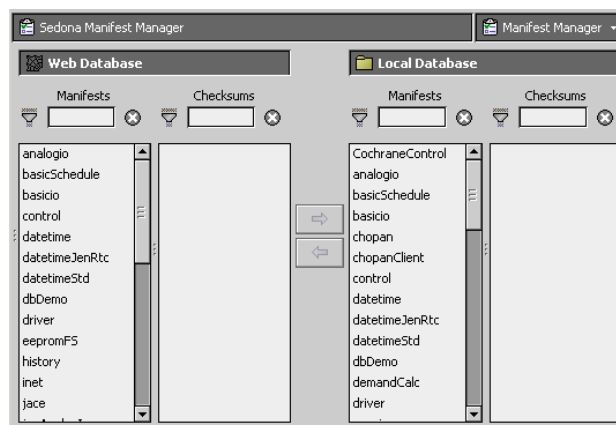


For complete details, including Workbench licensing information, see the *NiagaraAX Sedona Installer Guide* document.

Sedona Manifest Manager

The **Sedona Manifest Manager** is the tool you use to manage Sedona kit manifest files on your Sedona-enabled Niagara Workbench platform.

Figure A-5 Sedona Manifest Manager



For complete details, see the *Sedona Framework Manifest Manager - Engineering Notes* document.

