Technical Document

Niagara$^{AX-3.5}$ Z-Wave Driver Guide

Updated: March 24, 2015

Niagara^{AX} Z-Wave Driver Guide

Confidentiality Notice

Trademark Notice

Copyright and Patent Notice

Contents

Z-Wave Component Guides 5–37

PREFACE

Preface

This documents usage of the Z-Wave® (ZWave) driver for the NiagaraAX Framework.

Note: *The audience is considered to be knowledgeable in Z-Wave technology and is NiagaraAX Certified.*

The following main sections are in this document:

- This **`Preface`**, which provides brief descriptions of the major sections in this document below. Also included are a few Z-Wave terms and a Document Change Log.
- "Compatibility, Software Installation, and Jobsite Requirements" on page 1-1
  Explains the Z-Wave protocols and versions supported, including all supported Z-Wave command classes, as well as the NiagaraAX platform, software, and licensing requirements. Also included are *jobsite requirements*, explaining the need for a Z-Wave portable primary controller, the Z-Wave operation modes for the installed JACE, and special considerations about battery-powered devices.
- "Z-Wave Driver Quick Start" on page 2-1
  Provides several quick procedures for online station configuration to add a ZWaveNetwork, install the JACE on the Z-Wave network, and review dynamically created Z-Wave proxy points.
- "NiagaraAX Z-Wave Concepts" on page 3-5
  Provides concepts behind the Z-Wave driver, including all major components and views, including various screen captures.
- "ZWave Plugin Guides" on page 4-33
  Provides brief summaries of the different ZWave manager views, each with links back to the more detailed concepts section. Entries are used in NiagaraAX context-sensitive help "On View".
- "Z-Wave Component Guides" on page 5-37
  Provides brief summaries of the different ZWave components, some with links back to the more detailed concepts section. Entries are used in NiagaraAX context-sensitive help "Guide On Target".

Z-Wave terms

The following list of terms and abbreviations is specific to Z-Wave usage in NiagaraAX, and covers entries used in this document. For terms found in other Z-Wave related publications, refer to documents published by Zensys. See "Z-Wave reference documents" on page 1-1.

Note: *For general NiagaraAX terms, see the Glossary in the* User Guide*.*

battery-powered device To conserve batteries, most battery-powered Z-Wave slave devices "sleep" most of the time, and thus can be considered "non-listening" nodes. Whereas, line-powered slave devices are considered "always listening" nodes. Z-Wave driver support for battery-powered devices requires special considerations. See "Battery-powered (non-listening) device considerations" on page 1-5.

FLiRS (Frequently Listening Routing Slave) - A special battery-powered Z-Wave device that functions as a routing slave. See "Frequently Listening Routing Slave (FLiRS) device considerations" on page 1-5.

group Multiple Z-Wave devices can be controlled with a single command when placed in a group. Groups are named Group 1, Group 2, and so on.

node A Z-Wave device installed on a Z-Wave network. Each node must have a unique "node ID", an integer value from 1 to 223. Most nodes are slave devices; one or more nodes is a controller device.

primary controller The main device used to set up a Z-Wave network. There can be only one primary controller, and it must be used to add or delete devices. A primary controller can be a portable device like a hand-held remote, or a static controller (permanently installed and never removed).

routing slave A Z-Wave device that has all slave functionality plus can hold several routes. Routes are assigned by a controller.

scene A scene is a pre-defined combination of actions that occurs from one command. Scenes are typical to lighting control. However, if other devices support scenes, a scene may include other combinations of Z-Wave device control. For example, a scene may close the blinds, turn on inside lights to a pre-set level, and activate an outdoor motion sensor. Multiple scenes can be defined (Scene 1, Scene 2, etc.).

secondary controller A Z-Wave network supports multiple controllers, where a secondary controller may be added. A secondary controller cannot be used to add or delete devices; however, it may be able to replicate the configuration of the primary controller.

slave A Z-Wave device that can transmit information to other Z-Wave nodes only when requested to do so. Slave devices usually have physical inputs and outputs for monitor and control, and typically are the most numerous nodes on a Z-Wave network.

SUC (Static Update Controller) A secondary controller defined as the SUC (Static Update Controller) receives updates from the primary controller when node changes occur on the Z-Wave network. Other controllers on the network can individually request the SUC for network updates. There can be only one SUC on a Z-Wave network.

SIS (SUC ID Server) An SUC (Static Update Controller) can further be configured as a node ID server, allowing all other nodes to include/exclude nodes. The SIS automatically becomes the primary controller when enabled. To maintain consistency, all node ID allocations are maintained by the SIS. There can be only one SIS on a Z-Wave network.

Document Change Log

Updates (changes/additions) to this *NiagaraAX Z-Wave Driver Guide* document are listed below.

- Updated: March 24, 2015
  Added Z-Wave legal information to Copyright and Patent Notice information.
- Updated: November 15, 2010
  Content changes to describe added functionality in the (`zwave`) driver module starting in build 3.5.33.1, for official release. Document changes include more entries added in this preface's collection of "Z-Wave terms", and in the "Compatibility, Software Installation, and Jobsite Requirements" section, an updated section "Battery-powered (non-listening) device considerations" on page 1-5, followed by a new section "Frequently Listening Routing Slave (FLiRS) device considerations".
  In the main "NiagaraAX Z-Wave Concepts" section, new ZWaveNetwork properties "Auto Associate", "Auto Route", and "Read Back Time" are described in the section "Other interface related properties" on page 3-9, and a new "flirs" tuning policy is noted in "ZWaveNetwork tuning policy notes" on page 3-10. New device-level properties "Max Fail Until Device Down" and "Read Device On Event Update" are described in the section "ZWaveDevice common properties" on page 3-15, along with new device "Node Info" properties "Is Listening", "Optional Functionality", "Sensor250ms", and "Sensor1000ms". A new (and related) "Request Protocol Information" action is also described in "ZWaveDevice actions" on page 3-19. The "About ZWave proxy points" section "ZWaveProxyExt properties" on page 3-21 was updated to describe new properties "Rewrite Emergency Cmds", "Age", and "Next Wake Up In". Various figures showing property sheet screen captures were updated to reflect new properties.
- Updated: November 3, 2010
  Minor typo corrections, scene descriptions were modified slightly to emphasize lighting.
- Publication: August 17, 2010
  Initial document.

Chapter 1

Compatibility, Software Installation, and Jobsite Requirements

Currently, this section has the following main subsections:

Compatibility

The NiagaraAX ZWave driver has the following compatibility criteria:

NiagaraAX platform compatibility

Note: *NiagaraAX-3.5 or later is required.*

The ZWave driver will function on all NiagaraAX platforms that support JACE option cards or RS-232 serial communications.

- If a a QNX-based JACE, that is a JACE-2, 6, 7, or XPR (M2M JACE) series, a Z-Wave option card can be installed and used as the wireless Z-Wave interface.
- Alternatively, a serially connected, external, third-party Z-Wave gateway device can be used as the wireless Z-Wave interface. Table 1-1 lists gateway models that were tested with the ZWave driver.

Table 1-1 *Tested third-party Z-Wave serial gateways*

Manufacturer	Model	Description
Advanced Control Technologies, Inc.	ZCX101 (US)	HomePro Z-Wave Computer Interface
	ZCS201 (EU)	
Cooper Wiring Devices	RF232 (US)	Serial Interface Programming Module
HomeSeer	Z-Troller (US)	Serial Interface Programming Module

Z-Wave compatibility

The ZWave driver was developed against the Z-Wave Command Class Specification, Version 8, and also the Z-Wave Device Class Specification, Version 18. Specification documents for both are included in the Z-Wave reference documents section that follows. The driver was tested with Z-Wave Gateways running versions 2.24 and 2.74 of the static controller library.

Z-Wave reference documents

The following documents, listed by Zensys document ID number and title, may be referenced in this driver document.

1. SDS11060-8 *Z-Wave Command Class Specification*
2. SDS10242-18 *Z-Wave Device Class Specification*
3. SDS10243-8 *Z-Wave Protocol Overview*
4. INS102044-5 *Z-Wave Node Type Overview and Network Installation Guide*

Also see the section "Z-Wave command classes supported" on page 1-2.

Z-Wave command classes supported

When replicating (receiving) a Z-Wave network as a static secondary controller, or when adding Z-Wave devices with the JACE operating as the SUC or SIS node, the driver automatically creates all Niagara components and proxy points. These map to the Z-Wave command classes that each device supports.

The NiagaraAX ZWave driver supports the command classes listed in Table 1-2, which also provides links to related reference topics in this document.

Note: *Command classes listed with an asterisk (*) have been implemented, but have not yet been tested against a Z-Wave device that also supports that command class.*

Table 1-2 *Supported Z-Wave command classes*

Z-Wave Command Class	Ver.	Component under ZWaveDevice?	Proxy points under ZWaveDevice?
Alarm Sensor	V1	None.	1 to 6 BooleanPoints.
Alarm Silence	V1	None.	None, action added to points above.
All Switch	V1	switchAll (CcSwitchAll)	None.
Application Status *	V1	applicationStatus (CcApplicationStatus)	None.
Association	V1	associations (ZWaveAssociationDeviceExt), group*n* (ZWaveAssociationGroup)	None.
Association *	V2		
Basic	V1	None.	1 “basic” (type varies by device class)
Basic Tariff Information	V1	basicTariffInfo (CcBasicTariffInfo)	Yes, 1 or 2 Proxy points.
Battery	V1	None.	1 “batteryLevel” and 1 “batteryLow”.
Battery Sensor	V1	None.	1 “batterySensor”.
Binary Switch	V1	None.	1 “switch” BooleanWritable.
Binary Toggle Switch	V1	None.	1 “switchToggle” BooleanWritable
Clock	V1	clock (CcClock)	None.
Configuration	V1	configuration (ZWaveConfigDeviceExt)	None.
Door Lock *	V1	None.	—
Energy Production *	V1	None.	4 NumericPoints.
Hail	V1	No. (fires “Hail” topic on parent device).	None.
HRV Status *	V1	hrvStatus (CcHrvStatus)	Yes, from 1 to 7 Proxy points.
HRV Control *	V1	hrvControl (CcHrvControl)	Yes, from 1 to 7 Proxy points.
Indicator	V1	indicator (CcIndicator)	None.
Lock	V1	None.	1 “lock” Boolean Writable.
Manufacturer Specific	V1	manufacturerSpecific (ZWaveManufacturerSpecific)	None.
Meter	V1	None.	1 “meterValue” NumericPoint.
Move To Position Window Covering *	V1	None.	1 “mtpWindowCovering” NumericWritable.
Multi Channel Association	V2	associations (ZWaveAssociationDeviceExt), group*n* (ZWaveAssociationGroup) plus additional assocInEp(*n*) slots	None.
Multi Instance	V1	multiInstance (CcMultiInstance), where V2 exposes additional properties	None.
Multi Channel	V2		
Multilevel Sensor	V1	None.	1 “multiLevelSensor” NumericPoint, where V2 has more units/precision facets
Multilevel Sensor	V2		
Multilevel Switch	V1	None.	1 “switch” NumericWritable.
Multilevel Switch	V2		
Node Name & Location	V1	Naming (ZWaveNames)	None.
Protection	V1	protection (CcProtection)	None.
Pulse Meter	V1	None.	1 “pulseCount” NumericPoint.

Z-Wave Command Class	Ver.	Component under ZWaveDevice?	Proxy points under ZWaveDevice?
Scene Activation	V1	Scenes (ZWaveSceneNetworkExt), *scenes* (ZWaveNetworkScene)	None.
Scene Actuator Configuration	V1	*userAssigned* (ZWaveNetworkSceneConfig), scene*n* (ZWaveSceneActuatorConf)	None.
Thermostat Fan Mode	V1	None.	1 “fanMode” EnumWritable.
Thermostat Fan State	V1	None.	1 “fanState” EnumPoint.
Thermostat Mode	V1	None.	1 “mode” EnumWritable, where V2 has more possible modes.
Thermostat Mode	V2		
Thermostat Setpoint	V1	None.	1 to 16 NumericWritable points, depending on the setpoints supported by the device.
Thermostat Setpoint	V2		
Time	V1	time (CcTime).	None.
User Code *	V1	users (ZWaveUserDeviceExt), user*n* (UserInfo).	None.
Version	V1	version (ZWaveVersion).	None.
Wake Up	V1	wakeUp (CcWakeUp). V2 exposes additional properties.	None.
Wake Up	V2		

License requirements

To use the NiagaraAX ZWave driver, you must have a target NiagaraAX host (JACE) that is licensed with the “`zwave`” feature, as well as the “`serial`” feature. In addition, the “`zwave`” feature may have other device limits or proxy point limits.

Software installation

From your PC, use the Niagara Workbench 3.5.*nn* or higher installed with the “installation tool” option (checkbox “This instance of Workbench will be used as an installation tool”). This option installs the needed distribution files (*.dist* files) for commissioning various models of remote JACE platforms. The dist files are located under your Niagara install directory under a “sw” subdirectory. For more details, see “About your software database” in the *Platform Guide*.

Apart from installing the 3.5.*nn* or higher version of the Niagara distribution in the JACE, make sure to also install the *zwave* module too, plus any modules shown as dependencies. For more details, see “About the Commissioning Wizard” in the *JACE NiagaraAX Install and Startup Guide*.

See the next section Z-Wave jobsite requirements for additional jobsite requirements.

Z-Wave jobsite requirements

You must install the JACE on the Z-Wave network using a “Z-Wave portable primary controller”. The portable primary controller is also used to install other Z-Wave devices on that network.

First, you must decide on the Z-Wave operating mode for the JACE. This may depend on the capabilities of that primary controller, as well as if battery-powered Z-Wave slave devices will be installed.

The following sections provide more background details:

- About a Z-Wave portable primary controller
- About JACE Z-Wave operation modes
- Battery-powered (non-listening) device considerations
- Frequently Listening Routing Slave (FLiRS) device considerations

About a Z-Wave portable primary controller

A Z-Wave portable primary controller is a device that you need to install (or remove) Z-Wave devices as “nodes” on a Z-Wave network. Internally, it maintains the network management for the network, including all available node ID (addressing) information.

As a portable device, you carry the primary controller around the jobsite, such that you can key in an operation (such as adding a node) while physically next to that device. Most Z-Wave devices have a button or switch that you press to coordinate this operation from the “slave” side.

Z-Wave primary controllers also typically allow you to engineer other facets of a Z-Wave network too, such as adding "groups" and "scenes". However, it is usually faster and simpler to do this from the Niagara station interface to the Z-Wave network.

About JACE Z-Wave operation modes

As a secondary controller, the JACE can be installed to operate in *one* of these three Z-Wave modes:

- Secondary Static Controller
  Or simply "Static Controller". This mode is *not recommended* if any battery-operated Z-Wave slave devices are installed. In addition, either of the other two operating modes provide better overall performance and ease of operation.
- Static Update Controller (SUC)
  This is a better mode, and suitable if battery-powered slave devices are installed.
- SUC ID Server (SIS)
  This may be the best mode, as it provides all SUC functionality, plus additional flexibility.

Secondary Static Controller

Devices on the Z-Wave network should be installed *before* adding the JACE operating in this mode.

- The portable primary controller must support replication — to send its network configuration to a secondary controller.

Note: *In this mode, if battery-powered devices are used, and new devices are added after the battery-powered devices are modeled in the JACE station, the JACE will no longer receive "wake up" notifications from the battery-powered nodes. This happens because subsequent replications* change *the JACE's interface node ID. For related details, see "Battery-powered (non-listening) device considerations".*

Static Update Controller (SUC)

When a controller is configured as an SUC, the primary controller automatically sends network updates to the SUC, for example, when a new node is added to the network. Therefore the new node automatically appears in the SUC's (JACE's) topology map.

Other controllers in the network may individually request the SUC for network updates. If no SUC is present, the primary controller is responsible for updating all controllers in the network. Typically, this will be a manual process for the installer.

The Z-Wave network should *not* already be installed when installing the JACE to operate in this mode. If already installed, you must use the primary controller to uninstall all Z-Wave nodes first.

There can be only *one* SUC in a Z-Wave network (the JACE)—the first device installed.

The portable primary controller must support:

- Replication — to send its network configuration to a secondary controller (JACE).
- Transfer of primary controller role - to send this role to the JACE.
- Transfer back of the network configuration - to receive this information back from the JACE.

The primary controller must remain on the jobsite, dedicated to this specific job.

SUC ID Server (SIS)

When a SUC is also configured as node ID server (SIS), it enables all other controllers to include/exclude nodes. The SIS automatically becomes the primary controller in the network when enabled. To maintain consistency, all node ID allocations are maintained by the SIS.

The Z-Wave network should *not* already be installed when installing the JACE to operate in this mode—if already installed, you must uninstall all Z-Wave nodes first.

There can be only *one* SIS in a Z-Wave network (the JACE)—the first device installed.

The portable primary controller must support all items listed in the "Static Update Controller (SUC)" section, plus:

- Be able to operate as a Z-Wave "Inclusion Controller".

After the job is installed (if needed), the primary controller can be taken from the jobsite, "cleared", and used to install other jobs.

Battery-powered (non-listening) device considerations

Support of battery-powered Z-Wave devices requires special considerations. These devices "sleep" the majority of the time, "waking up" on a defined interval, or when the device detects an event.

As previously mentioned, to properly support battery-powered nodes, the JACE should be configured as an SUC or SIS. This will keep the JACE's Z-Wave interface node ID from changing. It also allows notification when a Z-Wave device has been installed on the network. The JACE can "discover" a battery-powered device when it is installed, since it is in an awake state at that time.

The "Wake Up" command class of a battery-powered device is configured with the node ID of the device that it will send wake-up notifications. If the JACE is configured as an SUC or SIS, the node ID of the JACE's Z-Wave interface is used—otherwise, a Z-Wave broadcast address is used.

If the battery-powered device also supports *associations*, the driver automatically adds the JACE to the devices' association groups. Typically, this causes the JACE to be notified via the "Basic" command class report when the device detects an event.

There seems to be some inconsistency between vendors and devices in the way battery-powered devices react when they detect an event. Some devices send a Wake Up command class wake up notification message, while others only send a Basic command class report to the association group members.

The Z-Wave driver does support posting changes to the Configuration and Associations command classes, so that those changes will be processed on the next wake up. Other command classes that the battery-powered device may also support require a coordinated *manual* wake up of the device, prior to making changes.

The "Naming" command class is an example. If you wish to change the Z-Wave name or location of a battery-powered device, it requires that the device is awake when you make the change.

When the driver evaluates a device as asleep, it sets the device's stale flag. When the device is detected as awake, the stale flag is cleared.

The driver does not ping the device to determine device up/down status. Its down flag will never be set, unless the device also implements the WakeUp command class. If it does, the device is marked down only if a WakeUp Notification message is not received at the interval specified in its WakeUp command class.

Frequently Listening Routing Slave (FLiRS) device considerations

FLiRS devices (Frequently Listening Routing Slaves) are battery-powered devices that are considered to be always listening. These devices rely on a special extended preamble, called a "beam", which allows a sleeping device to detect a frame within a very short wake-up interval.

The Z-Wave driver identifies these devices by either of the Sensor250ms or Sensor1000ms flags being set in the device's "NodeInfo" component. Such devices often include smoke detectors, or door or window monitors, where input values seldom change, but which still need to participate in message routing.

Because these devices are considered to be always listening, the driver assumes that it can send a message to one at any time. So the normal device ping and points polling still occurs, with the following exception:

- A special "**`flirs`**" tuning policy is automatically added to the ZWaveNetwork.
  This policy has a "MinReadTime" property with a default value of 1 hour. This property specifies the minimum time that must pass proxy point polls.With the MinReadTime set to 1 hour, proxy points in the device will not be polled any faster than once per hour.
  Note: *Points under a FLiRS device automatically have their tuning policy set to the "flirs" tuning policy.*

The main purpose is to poll these points at a much slower rate without consuming the "Slow" Poll Scheduler thread. If one of these points actually changed, the driver would be notified unsolicited, because that device would have been automatically *associated* to the JACE during the install process.

Chapter 2

Z-Wave Driver Quick Start

This section provides a collection of procedures to use the NiagaraAX ZWave driver to build a ZWaveNetwork with ZWaveDevices and proxy points. Like other NiagaraAX drivers, you do most configuration from special "manager" views and property sheets using Workbench.

Note: *A Z-Wave "portable primary controller" is required, along with a plan for the Z-Wave operation mode to be used by the JACE. In addition, there are licensing and software requirements for the driver. Please see "Compatibility, Software Installation, and Jobsite Requirements" on page 1-1 for more details.*

These are the main quick start subsections:

- Add and configure the ZWaveNetwork
- Install the JACE on the Z-Wave network
- Review ZWave proxy points and device child components

Add and configure the ZWaveNetwork

- Add the ZWaveNetwork
- Configure ZWaveNetwork communication parameters

Add the ZWaveNetwork

The ZWaveNetwork is the top-level ZWave component in the station.

To add a ZWaveNetwork in the station

Step 1 Double-click the station's **Drivers** container, to bring up the **Driver Manager**.

Step 2 Click the **New** button to bring up the New network dialog. For more details, see "Driver Manager New and Edit" in the *Drivers Guide*.

Step 3 Scroll to select "`Z Wave Network`," number to add: `1` and click **OK**.

This brings up a dialog to name the network.

Step 4 Click **OK** to add the ZWaveNetwork to the station.

You should have a ZWaveNetwork named "ZwaveNetwork" (or whatever you named it), under your Drivers folder, showing a status of "`{fault}`" and enabled as "`true`."

After you Configure ZWaveNetwork communication parameters, status should change to "`{ok}`".

Configure ZWaveNetwork communication parameters

In the ZWaveNetwork property sheet you must define at least one parameter for communications.

To set the ZWaveNetwork communications parameters

To set the communications parameters for a ZWaveNetwork:

Step 1 Right-click the ZWaveNetwork and select **Views > Property Sheet**.

The **Property Sheet** appears.

Step 2 Expand the **Comm Plug In** slot, then its **Serial Port Config** slot.

Set the properties for the JACE serial port used, starting with **Port Name**. Properties are as follows:

- Port Name: `none` — Enter the JACE RS-232 port being used, like COM2 or COM3.
- Baud Rate: `Baud9600` — Or choose different from selection list.
  Note: *For a Z-Wave JACE option card, set this to* `Baud115200` *(115,200 baud). The remaining serial port properties below are left at defaults (8, 1, N, none).*

- Data Bits: `Data Bits8` — Or choose different from selection list.
- Stop Bits: `Stop Bit1` — Or choose different from selection list.
- Parity: `None` — Or choose different from selection list.
- Flow Control Mode: `none` — Or choose different using checkbox.

Note: *If an external Z-Wave serial gateway is configured differently from the defaults above, adjust the baud rate, data bits, stop bits, parity, and flow control settings as necessary to match.*

Step 3 Click the **`Save`** button.

For further details on the ZWaveNetwork, see "About the ZWaveNetwork" on page 3-7.

Install the JACE on the Z-Wave network

The Z-Wave driver is designed to act as a Z-Wave secondary controller, operating in one of three modes: as a Static Controller, a Static Update Controller (SUC), or a SUC ID Server (SIS) on the Z-Wave network. If possible, it is *highly recommended* that you configure the network so that the JACE *is an SIS or SUC*. This provides the best overall performance and ease of installation.

Note: *If battery-powered devices are going to be used on the Z-Wave network, consider this SUC or SIS operation as a requirement. For related details, see "About JACE Z-Wave operation modes" on page 1-4.*

The two main methods use these different install procedures:

- To install the JACE as an SUC or SIS (recommended)
- To install the JACE as a Static Controller

To install the JACE as an SUC or SIS

The portable primary controller used to install Z-Wave devices *must support* SUC and/or SIS functionality. The JACE must be the *first Z-Wave device installed*, in order for it to operate as an SUC or SIS. It is assumed that the JACE has already been configured, and it has a running station with a ZWaveNetwork.

Use the following procedure to install the JACE on the Z-Wave network as an SUC or SIS device.

Step 1 Uninstall any Z-Wave devices previously installed, using the portable primary controller.

Step 2 It is recommended you reset the portable primary controller back to its factory defaults.

This should remove any network/node information, ensuring that the JACE will be the *first* node added (as a result of transferring the primary role to it, later in this procedure).

Step 3 Open the property sheet of the **`ZWaveNetwork`**.

The property "**`Configure As SIS`**" is used to select the JACE to act as either an SIS or SUC. The default value is `true` (SIS). If SUC operation is needed, change to `false`, and **`Save`**.

Step 4 Transfer the primary controller role to the JACE.

1. Double-click the network, to bring up the device manager (**`ZWave Device Manager`**).
2. Prepare the portable primary controller to initiate a replicate (send), to shift the primary role.

Note: *Although it may seem the primary controller has nothing to send on replication (it was cleared in Step 2), in fact this results in the JACE being added as the first node, with the node ID 2.*

3. In the **`ZWave Device Manager`**, click the button **`ReplicateReceive`**.

   Within a few seconds, from the primary controller, complete the replication send / primary role shift, as started in substep 2.

4. After this replication completes, the "Home ID" property of the **`ZWaveNetwork`** should match that of the primary controller, and its "Interface Node Id" property should be 2.

Step 5 Transfer back to the primary controller. If SIS operation has been selected, this make the portable primary controller a "Z-Wave Inclusion Controller".

1. Using the portable primary controller, initiate a replicate receive.
2. In the **`ZWave Device Manager`**, click the button **`TransferPrimary`**.

   (This substep must be done within 30 seconds of the previous substep.)

3. This should leave the JACE operating in the desired SIS or SUC role.

Now you can use the portable primary controller to install other Z-Wave devices. As each device is installed, the JACE is automatically notified. The Z-Wave driver automatically adds the appropriate ZWaveDevice component, including child components and proxy points, according to the device's supported Z-Wave command classes.

Note: *By operating in the SIS or SUC mode, this eliminates the need for any additional replications to the JACE.*

For a summary of how all the command classes are modeled, including the dynamically added components, see "Z-Wave command classes supported" on page 1-2.

To install the JACE as a Static Controller

Note: *Z-Wave driver versions prior to 3.5.25.7 only supported this mode of operation. However, it is recommended that the JACE be configured as an SIS or SUC instead. See "To install the JACE as an SUC or SIS". If these modes are not available, the JACE will still work as a Static Controller. However, issues with battery-powered devices are likely, as described in "Z-Wave jobsite requirements" on page 1-3.*

The Z-Wave network of devices should already be installed, using a portable primary controller. This primary controller must support replication to a secondary controller (in this case, the JACE). It is assumed that the JACE has already been configured, an it has a running station with a ZWaveNetwork.

Use the following procedure to install the JACE on the network, while replicating network information from the portable primary controller.

Step 1 Using the portable primary controller used to install Z-Wave devices on that network, prepare to initiate a replicate (send).

Step 2 Double-click the **`ZWaveNetwork`**, to bring up the device manager (**`ZWave Device Manager`**).

1. In the **`ZWave Device Manager`**, click the button **`ReplicateReceive`**.
2. After a second or two, while holding the Z-Wave portable primary controller near the JACE, initiate the replicate send. (You have up 30 seconds after clicking **`ReplicateReceive`**.)

   This automatically starts a discovery job that performs the following sequence:
   - Transfers the Z-Wave network topology from the Z-Wave portable primary controller to the Z-Wave secondary controller (JACE).
   - Adds a ZWaveDevice (ZWaveDevice or ZWaveThermostat) for each slave Z-Wave device that is discovered.
   - Reads the Z-Wave node information from each device, to determine the command classes supported by that device.
   - For each command class that the device supports (and that the driver also supports), adds an associated component under the device object, or proxy points,

Step 3 When the discovery job completes, the station should have all the components and points to provide monitor and control functions for each of the network's Z-Wave devices. For each device, the Z-Wave driver automatically adds the appropriate ZWaveDevice component, including child components and proxy points, according to the device's supported Z-Wave command classes.

Note: *If battery-powered devices are involved, it is recommended to force each battery-powered device to "wake up" 3 times for the JACE to learn all of its command classes. Typically, you do this by pressing a button or switch on a battery-powered device.*

For a summary of how all the command classes are modeled, including the dynamically added components, see "Z-Wave command classes supported" on page 1-2.

Review ZWave proxy points and device child components

As with device objects in other drivers, each ZWaveDevice 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 "**`ZWave Point Manager`**"). Use it to review (and if necessary), edit ZWave proxy points under any ZWaveDevice. For more details, see "ZWave Point Manager" on page 3-20.

Note: *Unlike point managers in almost all other drivers, you do not add proxy points using the* **`ZWave Point Manager`** *(no "New" or "Add" buttons, nor any "Learn mode" with a* **`Discover`** *button and pane).* All available proxy points are automatically created. *Of course you can add alarm or history extensions, link into other station logic, and "bind" to Px widgets in PxPages, just as with any other proxy points.*

In addition to proxy points, some data is modeled by dynamically added "child components" directly under each ZWaveDevice. As with proxy points, these child components will vary from one device to another, based on the Z-Wave command classes each device supports.

See the following procedures:

- To review ZWave proxy points
- To review ZWave device child components

To review ZWave proxy points

Once a ZWaveDevice is added, any available proxy points are already included. These will vary by the number and types of Z-Wave command classes that the device supports. You can get statuses and values back immediately, to review and use as needed.

To review (and edit) ZWave proxy points in a device:

Step 1 In the **`Device Manager`**, in the **`Exts`** column, double-click the `Points` icon in the row representing the device you wish to review points.

This brings up the **`ZWave Point Manager`**.

Step 2 Double-click any point for an **`Edit`** dialog.

To edit multiple points, hold the Ctrl key and click points to select, then click the **`Edit`** button.

In the **`Edit`** dialog you can (re)Name the point(s), edit or change point Facets, and change Tuning Policy Name, if needed. Additional properties are also available for editing, but are rarely if ever needed (Device Facets, for example).

Default point names are dependent on the command class, and typically use a "camel case" convention, starting with a lowercase letter—for example: "`switchToggle`" or "`dimmer`".

For more information, see "ZWave Point Manager" on page 3-20, "About ZWave proxy points" on page 3-21, and "ZWave proxy points by default name" on page 3-23.

To review ZWave device child components

Once a ZWaveDevice is added, all dynamic "child components" are already included. These will vary by the number and types of Z-Wave command classes that the device supports. You can expand the parent device to review and sometimes write values to properties in these child components. Or, as needed you can invoke actions, or link properties or actions into station control logic.

To review ZWave "child components" for any ZWaveDevice:

Step 1 In the Nav tree, expand the ZWaveDevice to see child components. All components listed under "Points" are dependent on the command classes supported.

Default names are sometimes reflective of the command class, and typically use a "camel case" convention, starting with a lowercase letter—for example: "`manufacturerSpecific`" or "`switchAll`".

Double-click any child component to see its properties (by themselves).

Or, you can go to the property sheet of a ZWaveDevice to see these same child components listed, and then expand any of interest.

Step 2 Right-click a child component to see any available actions.

For more information, see "About the ZWaveDevice" on page 3-14 and "ZWaveDevice child components" on page 3-18.

Chapter 3

NiagaraAX Z-Wave Concepts

This section provides conceptual details on the NiagaraAX ZWave driver and its components, including views. These are the main subsections:

- About Z-Wave to NiagaraAX Architecture
- About the ZWaveNetwork
- ZWave Device Manager
- About the ZWaveDevice
- ZWave Point Manager
- About ZWave proxy points
- About Z-Wave Scenes
- About Z-Wave Associations
- About Device Profiles

About Z-Wave to NiagaraAX Architecture

Z-Wave network architecture includes two main types of devices, or nodes; controllers and slaves. Essentially, controller nodes can calculate routes (and alternative routes) needed in wireless communications, whereas slave nodes primarily act as physical inputs and outputs on the network. The Z-Wave protocol supports a network with a maximum of 232 total nodes, including controller nodes and slave nodes.

Typically, a Z-Wave network has far fewer than 232 nodes, with the vast majority being slave nodes. Different subtypes of controller nodes and slave nodes exist (slave nodes can be "routing slaves", for example). At least one controller is required: a primary controller, typically a portable primary controller.

The functionality of any Z-Wave device is determined by the number and types of Z-Wave "command classes" it supports. An extensive set of command classes is available, most of which are also supported by the NiagaraAX Z-Wave driver.

In a NiagaraAX integration, the host (JACE) with Z-Wave option card (or external Z-Wave serial gateway) installs on the Z-Wave network as a controller node—minimally as a secondary "Static Controller". At installation time, the JACE can be promoted to a "Static Update Controller" (SUC) or "SUC ID Server" (SIS), providing that the (original) portable primary controller is capable. For related details, see "About JACE Z-Wave operation modes" on page 1-4.

Modeling Z-Wave in the NiagaraAX station

In the JACE station running the ZWave driver, a ZWaveNetwork models all slave nodes, where each node is represented by a ZWave device component (either a ZWaveDevice or ZWaveThermostat). No controller node is modeled. For example, no device component models the portable primary controller or the JACE's Z-Wave interface (apart from some ZWaveNetwork properties). However, the Z-Wave "node ID" value of 1 remains reserved for the Z-Wave portable primary controller.

Each ZWave device component, along with all its child components, is *automatically* (dynamically) added in the JACE station, once the JACE is installed on the Z-Wave network. Components and proxy points are automatically determined by a device's supported Z-Wave command classes. This varies from almost any other NiagaraAX driver, where some manual addition (or discovery and selection process), is needed using the network's "Device Manager" view, as well as in each device's "Point Manager" view.

Apart from this, the standard NiagaraAX network architecture applies, similar to other serial-based polling drivers. See "About Network architecture" in the *Drivers Guide* for more details. For example, real-time data is modeled using ZWave proxy points, which reside under ZWave devices, which in turn reside under a ZWaveNetwork container in the station's DriverContainer (Drivers).

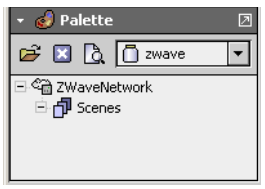
Hierarchically, the component architecture remains: network, device, points extension, points. The points extension is the only obvious "device extension" under a ZWave device—meaning there are no schedule or history device extensions.

Unique to a ZWaveNetwork are Z-Wave "scenes", which are modeled at the network level using a "Scenes" network extension. A scene defines some collection of actions to one or more Z-Wave devices that can be initiated using a single command. Special views on the network's "Scenes" extension and child "scene*n*" components provide easy methods to add new and edit (define) Z-Wave scenes.

Palette for ZWave driver

Although the `zwave` module has a palette, it contains only the **`ZWaveNetwork`** (and its frozen "**`Scenes`**" network extension). See Figure 3-1.

Figure 3-1 *Components in zwave palette*



All of the other various Z-Wave components (ZWave devices, components under devices, and proxy points) do not appear on the palette. Instead, the driver *dynamically adds* such components under the station's ZWaveNetwork, when installing the JACE on the Z-Wave network.

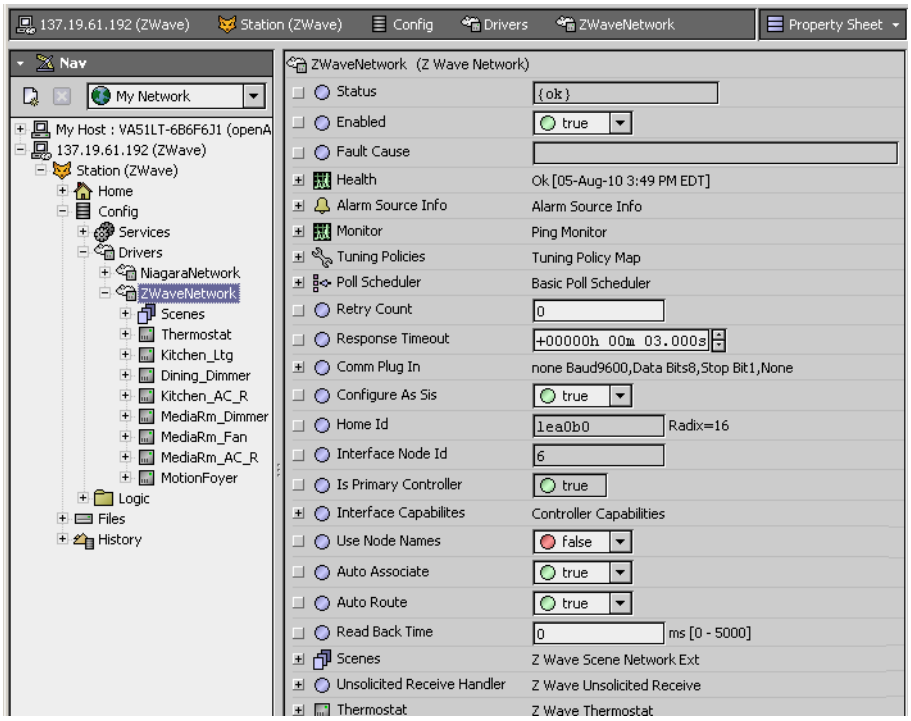
This occurs *automatically*, based upon "replications" from the Z-Wave portable primary controller, along with station command class queries to Z-Wave devices on the network. This simplifies component creation, and enforces proper component hierarchy.

Note: *As with most other NiagaraAX drivers, you rarely need to work from the palette. In this case, usefulness is limited to when first starting a Z-Wave integration—as an alternative to using the* **`Driver Manager`** *to add the new ZWave Network. (By copying from the palette, its default baud rate of 115,200 is pre-set to match that used by the JACE Z-Wave option card).*

About the ZWaveNetwork

The ZWaveNetwork is the top-level component for the ZWave driver in a station. On its property sheet (Figure 3-2), you configure specific settings under the "**`Comm Plug In`**" slot to allow communications to the physical Z-Wave interface (Z-Wave option card or external Z-Wave serial gateway). Properties also specify the station's Z-Wave operation mode. Status properties provide feedback on configuration.

Figure 3-2 *Property sheet of ZWaveNetwork*



The following sections provide more details on ZWaveNetwork properties and slots:

- ZWave-specific network slots
- Common ZWaveNetwork slots
- ZWaveNetwork actions

ZWave-specific network slots

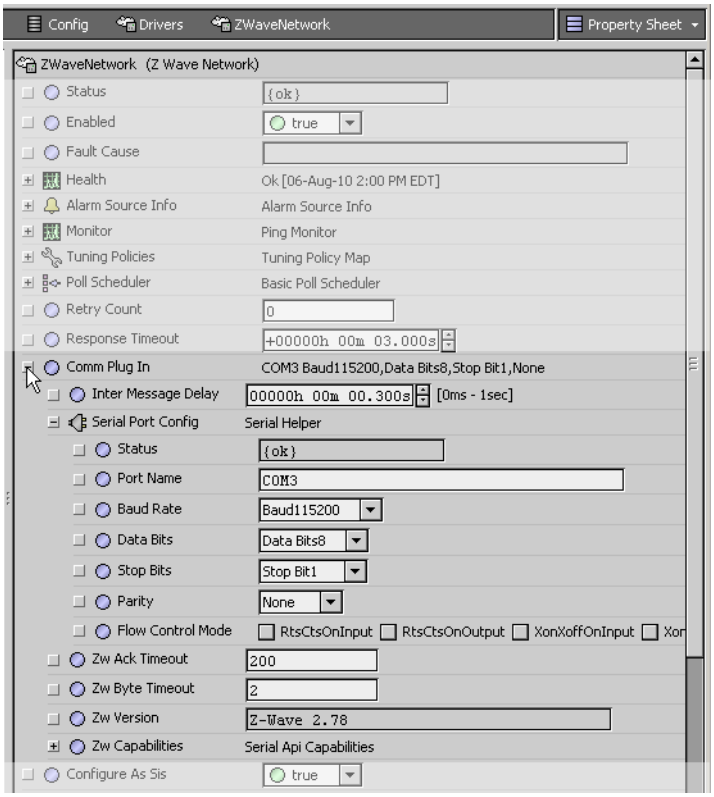
In addition to Common ZWaveNetwork slots, the ZWaveNetwork contains several container components and other configuration and status properties that relate directly to Z-Wave operation, as follows:

- Comm Plug In (Z-Wave communications parameters).
- Other interface related properties (mostly Z-Wave interface related).
- ZWave Scenes network extension (for Z-Wave scenes).

Comm Plug In

Communication parameters are set here to match the setup of the network controller's RS-232 port. Additionally, a few Z-Wave timeout parameters can be adjusted, and status properties reflect the API (application programming interface) capabilities of the JACE's Z-Wave interface.

Figure 3-3 *Comm Plug In expanded in ZWaveNetwork (using JACE Z-Wave option card)*



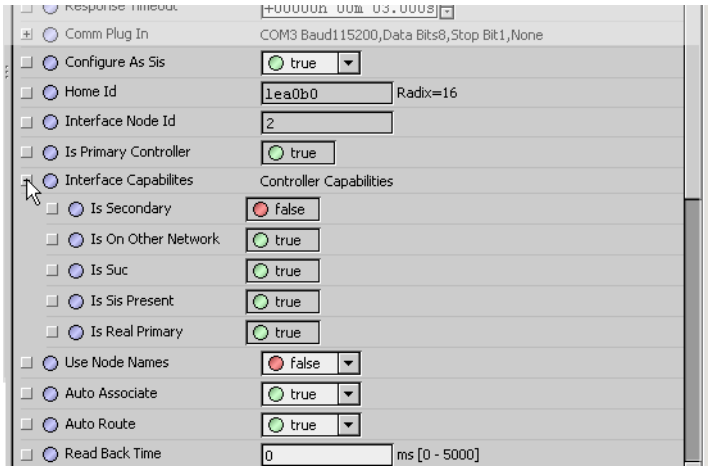
The following properties are found under the **`Comm Plug In`** component:

- **Inter Message Delay**
  Some installations require a "pause" between adjacent messages to allow time for the Z-Wave interface to "catch up" with its own internal processing. This has the effect of adding time between polls, so this number should be kept as small as possible. Normally, a setting of "0.0" will work.
- **Serial Port Config**
  Communications parameters are set here to match the set up of the Z-Wave serial gateway's RS-232 port. See "To set the ZWaveNetwork communications parameters" on page 2-1.
- **Zw Ack Timeout**
  Number of 10 millisecond ticks that the Z-Wave gateway waits for an ACK before timeout.
- **Zw Byte Timeout**
  Number of 10 milliseconds ticks that the Z-Wave gateway waits for a new byte received, once a received frame has been detected, before timeout.
- **Zw Version**
  Serial API version of the Z-Wave gateway.
- **Zw Compatibility**
  Contains a number of read-only status properties that reflect serial API functions supported.

Other interface related properties

Among ZWave-specific network slots, some Z-Wave interface-related properties are further down in the network's property sheet (Figure 3-4).

Figure 3-4 *Interface related properties in the ZWaveNetwork*



These properties include the following:

- **Configure As Sis**
  When set to true (default), the JACE attempts to establish the interface as the SIS for the Z-Wave network. Otherwise, it attempts to establish the interface as the SUC for the Z-Wave network.
- **Home Id**
  (read-only) The 24 bit Home ID for this Z-Wave network. This is automatically initialized during replication from a Z-Wave primary controller.
- **Interface Node Id**
  (read-only) The node ID of the Z-Wave interface (JACE Z-Wave option card, or external Z-Wave serial gateway). This is automatically initialized upon replication from a Z-Wave primary controller. If the JACE has been correctly installed as an SIS or SUC, this value should be `2`.
- **Is Primary Controller**
  (read-only) If true, indicates that the interface is acting as the primary controller for the network. This is an indication that the interface is acting as an SIS.
- **Interface Capabilities**
  Contains a number of read-only status properties that reflect the Z-Wave interface's operating mode. If operating as an SIS, all properties should be true except "Is Secondary", as shown in Figure 3-4. The "Is Secondary" property is true only if the JACE is not installed as an SIS or SUC, but instead a Static Controller. For related details, see "About JACE Z-Wave operation modes" on page 1-4.
- **Use Node Names**
  Default is false. If set to true, the driver automatically creates a display name for each device that also supports the "Naming" command class (Node Name and Node Location). The display name is a combination of the device's location and the name strings, along with node ID number.
  For example if device (node 7) named "Kitchen_AC_R" has a Node Name of "CoffeePot" and Node Location of "Kitchen", and this property is set true, it will appear listed in all views as:
  `Kitchen_CoffeePot_7` (instead of `Kitchen_AC_R`)
  If present, display names are used throughout views and dialogs instead of component names.
  Note: *If set true, any device that does not support the Naming command class appears instead using a combination of its "specificDeviceClass" and node ID number. For example a device (node 3) named "Thermostat" might instead appear as "*`thermostatGeneralV2_3`*". Find a device's specific device class string under its "Node Info" container slot.*
- **Auto Associate**
  If true (the default), the driver automatically creates an association to the JACE for all association groups supported by a device. This causes the data that is sent as the result of the association group event to update in the JACE, without waiting for the device to be polled.
- **Auto Route**
  If true (the default) the driver uses an "auto-route transmit option" when sending messages. This option causes the Z-Wave interface / option card to try transmitting messages via other repeater nodes

if communications fails. Be careful about changing this property, unless you are sure all nodes can be accessed via direct communications.

- **Read Back Time**
  Specifies the amount of time, in milliseconds, after a Z-Wave writable point sends a write message to a device that it triggers a subsequent read message to be sent to the device (to update that point's read value). Setting the value to 0 disables this feature.

ZWave Scenes network extension

Among ZWave-specific network slots is the "Scenes" network extension, used for creating, editing, and activating Z-Wave scenes. For more details, "About Z-Wave Scenes" on page 3-27.

Common ZWaveNetwork slots

The ZWaveNetwork component includes the typical collection of slots and properties as most other network components. For general information, See "Common network components" in the *Drivers Guide*. The following sections provide additional details:

- ZWaveNetwork status notes
- ZWaveNetwork monitor notes
- ZWaveNetwork tuning policy notes
- ZWaveNetwork poll scheduler notes
- ZWaveNetwork message handling properties

ZWaveNetwork status notes

As with most "fieldbus" drivers, the status of a ZWaveNetwork is either the normal "ok", or less typical "down" or "fault" (fault might result from licensing error, or if a non-existent COM port is assigned to Serial Port Config). The Health slot contains historical timestamp properties that record the last network status transitions from ok to any other status. The "Fault Cause" property further explains any fault status.

Note: *As in other driver networks, the ZWaveNetwork has an available "Alarm Source Info" container slot you can use to differentiate ZWaveNetwork alarms from other component alarms in the station. See "About network Alarm Source Info" in the Drivers Guide for more details.*

ZWaveNetwork monitor notes

The ZWaveNetwork's monitor routine verifies child ZWaveDevice component(s)—the "pingable" devices in the ZWave driver. The default ping frequency is every 5 minutes, and is adjustable. For general information, see "About Monitor" in the *Drivers Guide*.

ZWaveNetwork tuning policy notes

The ZWaveNetwork has the typical network-level Tuning Policy Map slot with a single default Tuning Policy, as described in "About Tuning Policies" in the *Drivers Guide*. Typically, only a single "Default Policy" exists, however, you can add new tuning policies (duplicate and modify) as needed.

If you have multiple tuning policies, you can assign Z-Wave proxy points as needed to different ones. Under any proxy point's **`Proxy Ext`** properties, in its **`Tuning Policy Name`** property, select the desired tuning policy.

Note: *If one or more FLiRS (Frequently Listening Routing Slave) devices are found on the network, the driver automatically adds* another *tuning policy named "***`flirs`***". Proxy points under such a device automatically use that tuning policy. For related details, see "Frequently Listening Routing Slave (FLiRS) device considerations" on page 1-5.*

ZWaveNetwork poll scheduler notes

The ZWaveNetwork has the typical Poll Scheduler slot, as described in "About poll components" in the *Drivers Guide*. It enables/disables polling, determines fast/normal/slow poll rates, and maintains statistics about proxy extension polls.

By default, a newly created ZWave proxy point uses the "Normal" poll rate. If needed, you can assign proxy points to different poll rates. Under any proxy point's **`Proxy Ext`** properties, select the rate in its **`Poll Frequency`** property.

ZWaveNetwork message handling properties

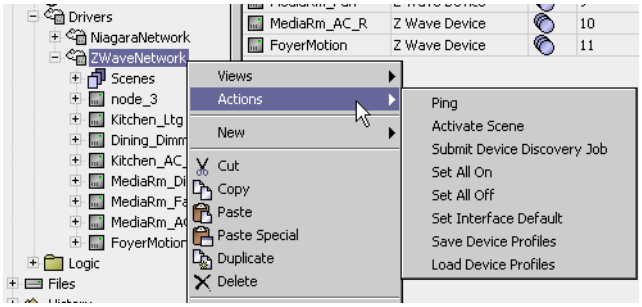
The ZWaveNetwork has several "message handling" network-level properties common among serial drivers, described separately as follows:

- **Retry Count**
  How many retries the communications handler will try to send a Z-Wave message if the initial attempt gets no response.
- **Response Timeout**
  Specifies the maximum time to wait for a response to a Z-Wave message once sent. If a response is not received before this timeout, the message is resent up to "Retry Count" times, each of which waits for this timeout period.
- **Unsolicited Receive Handler**
  (near bottom of property sheet) Contains a child "Unsolicited Message Count" property that tallies all unsolicited messages sent by the Z-Wave interface. For informational use only.

ZWaveNetwork actions

The ZWaveNetwork has a several available actions, as shown in Figure 3-5.

Figure 3-5 *Actions on ZWaveNetwork*



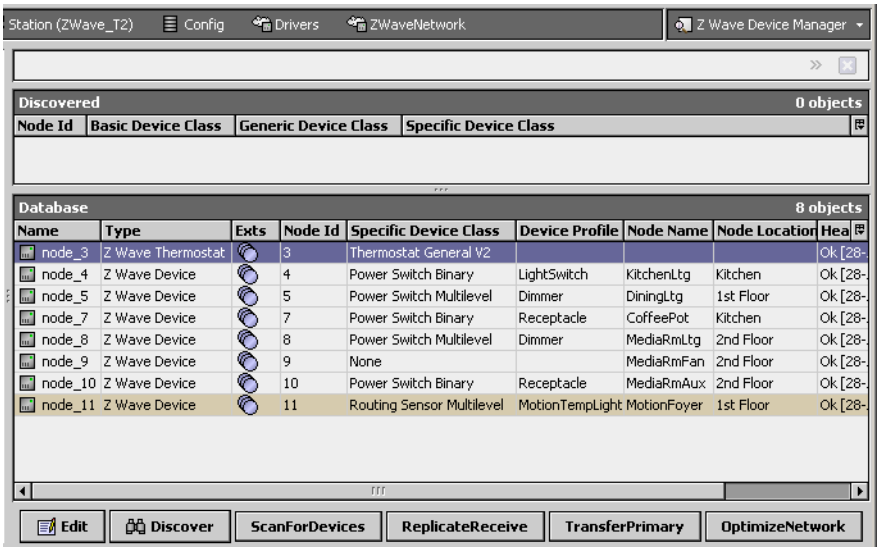
Actions on the ZWaveNetwork are described as follows:

- **Ping**
  Force a ping message to be sent to the Z-Wave interface.
- **Activate Scene**
  Produces a popup Activate Scene dialog, in which you specify a scene to activate (by scene number). This multicasts an activate scene message on the Z-Wave network.
- **Submit Device Discovery Job**
  Initiates a Z-Wave device discovery (same as a Discover from the **`ZWave Device Manager`**).
- **Set All On**
  Broadcasts a Z-Wave "All On" message on the network. Devices that support the `All Switch` command class respond as defined in their "switchAll" component. For reference details, see "zwave-CcSwitchAll" on page 5-41.
- **Set All Off**
  Broadcasts a Z-Wave "All Off" message on the network. Devices that support the `All Switch` command class respond as defined in their "switchAll" component.
- **Set Interface Default**
  Produces a popup confirmation dialog, which if answered "Yes" clears the Z-Wave network topology information from the JACE's Z-Wave interface (JACE Z-Wave option card or external Z-Wave serial gateway), setting this back to factory defaults.
  Note: *Use with caution, as this requires replication back from the primary controller again.*
- **Save Device Profiles**
  Forces saving device profiles back to the station's file system. See "About Device Profiles" on page 3-32.
- **Load Device Profiles**
  Forces loading of device profiles from the station's file system.

ZWave Device Manager

The ZWave Device Manager (Figure 3-6) is the default view for the ZWaveNetwork, and works similar to other device managers that support online device discovery (with differences noted below). See "About the Device Manager" in the *Drivers Guide* for general information.

Figure 3-6 *ZWave Device Manager is default view for ZWaveNetwork*

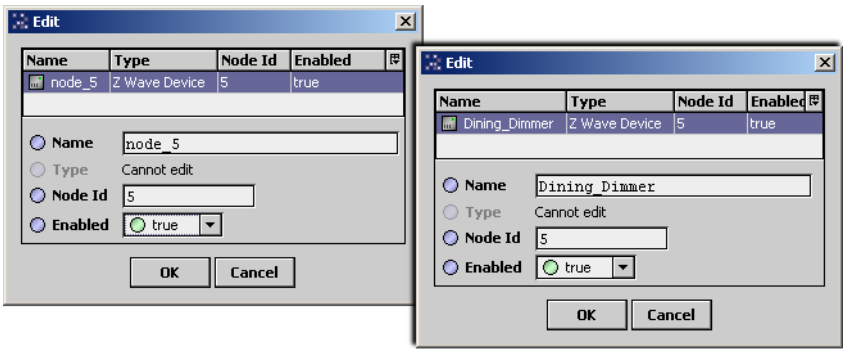


Note: *The ZWave Device Manager is unique from most Device Managers in that there are no "add" or "new" buttons—device components are added automatically as a result of "replication" or other network management functions. See the next section "Buttons (network management functions)" for details.*

From this view see all the Z-Wave slave devices on the network, along with their statuses. Figure 3-6 shows an example ZWave Device Manager view shortly after installing the JACE (as an SIS), where a "**ReplicateReceive**" (with resulting discovery) and "**TransferPrimary**" had been performed.

Double-click any device for an **Edit** dialog (Figure 3-7), or select multiple devices and click the **Edit** button, if needed.

Figure 3-7 *Edit dialog for ZWaveDevice*



Device defaults are like those shown in the Figure 3-7 (left) dialog above, where:

- Name is `node_n` (reflects the device's Z-Wave node ID), and
- Node Id is the Z-Wave device's node ID.
  Note: *Do not edit any device's Node Id! This is automatically determined upon replication.*

Typically you change a ZWaveDevice's *name*, as shown in Figure 3-7, right.

See "About the ZWaveDevice" on page 3-14 for more details on ZWave device components.

In this view, go to the points manager of any ZWaveDevice shown by double-clicking on its (Points) icon under the **Exts** column. See "ZWave Point Manager" on page 3-20.

Buttons (network management functions)

At the bottom of the ZWave Device Manager are buttons, which apart from the **Edit** button are:

Discover, ScanForDevices, ReplicateReceive, TransferPrimary, OptimizeNetwork

These provide network management functions, described as follows:

Discover

The **Discover** button launches a discovery job, which does the following sequence:

- Reads the node list from the Z-Wave interface (Z-Wave option card/ Z-Wave serial gateway).
- For each slave node, add a Z-Wave device to the station, if it doesn't already exist.
- For each device added to the station, request the "Node Information" data from the device.
  - Node Information includes a list of command classes that the device supports.
- For each command class that the device supports and that the driver also supports:
  - Create one or more components and/or control points (proxy points) to model the data for the command class.
  - If the components or proxy points already exist, it just updates the data within the component.
  - Sends a request to read the data contained within the command class.
- For each device that supports Association:
  - Reads the association information.
- Once discovery is complete, the Z-Wave network should be complete with devices and proxy points to monitor and control the slave Z-Wave devices

ScanForDevices

ScanForDevices performs the same sequence as a Discover, except it does not read associations.

ReplicateReceive

The **ReplicateReceive** button is to initiate a replication process. Replication is a Z-Wave process of copying the network topology information from the Z-Wave primary controller (that was used to install the Z-Wave network of devices) to the JACE's Z-Wave interface (JACE Z-Wave option card or external Z-Wave serial gateway).

The replication process is initiated clicking the **ReplicateReceive** button *just after* pressing the button sequence on the primary controller to *send* the replication messages.

This launches a "Replication Job" that initiates a sequence of RF messages to transfer the node table into the JACE's Z-Wave interface from the primary controller.

Once replication is complete, it *automatically* launches a discovery job (see "Discover"for details).

TransferPrimary

Some primary controllers have the ability to transfer "primary controller responsibility" during replication. In this case, the JACE's Z-Wave interface (JACE Z-Wave option card or external Z-Wave serial gateway) becomes the primary controller for the network. The user may do this if portable primary controller used to install the network is not left at the site (it may be used to install other sites).

The **TransferPrimary** button becomes active if the JACE's Z-Wave interface has been given the role of primary controller. Clicking this button initiates a replication process to transfer the network topology information and primary controller role back to a portable controller that can be used to add or remove Z-Wave devices to the network.

OptimizeNetwork

This button launches a job that sends messages to each device, instructing them to re-evaluate their return route to the JACE. You may wish to do this if you changed a device's physical location.

About the ZWaveDevice

A ZWaveDevice represents a ZWave slave node, and is mapped to a specific device by its unique Z-Wave node ID (Node Id) property.

The following main subsections provide more details:

- Types of ZWave device components
- ZWaveDevice common properties
- ZWaveDevice child components
- ZWaveDevice actions

Types of ZWave device components

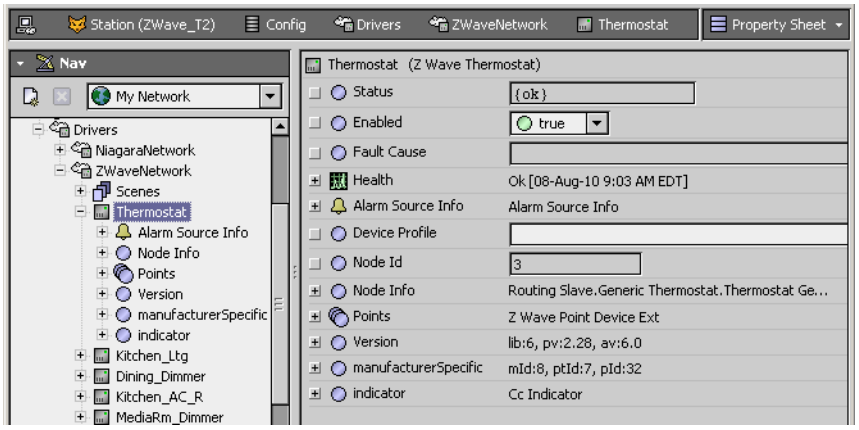
Although the term "ZWaveDevice" is used throughout this document when referring to *any* child device component under the station's ZWaveNetwork, technically there are *two* different device components that represent a Z-Wave slave device:

- ZWaveThermostat
- ZWaveDevice

ZWaveThermostat

A ZWaveThermostat component is dynamically created for any discovered slave device that implements a "thermostat type" generic or specific device class. Figure 3-8 shows an example ZWaveThermostat's property sheet.

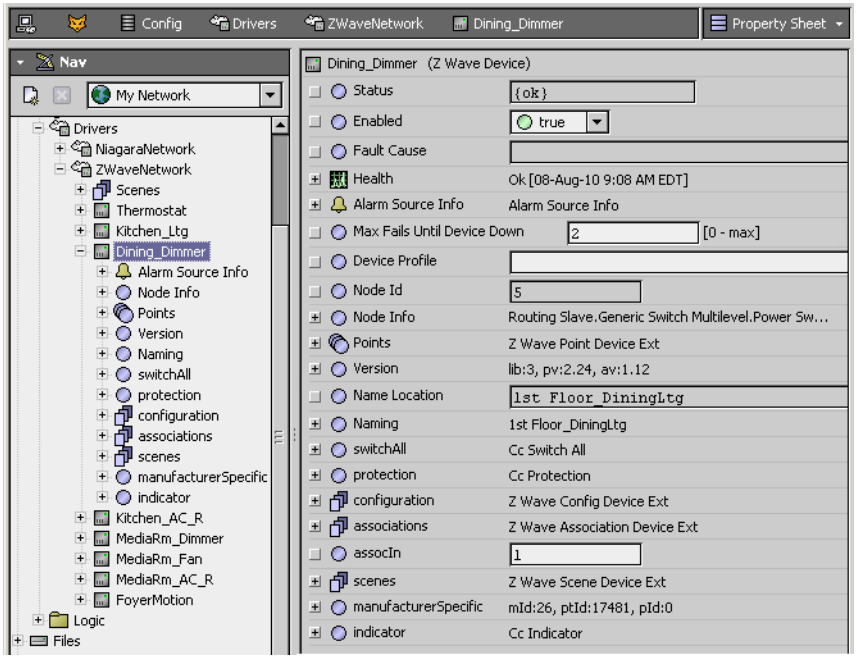
Figure 3-8 *Example ZWaveThermostat component, property sheet*



ZWaveDevice

A ZWaveDevice component is dynamically created for any discovered slave device using a generic or specific device class that is *not a "thermostat type"*. For example, a "switch binary", "switch multilevel" (dimmer), "sensor multilevel", and so on. Figure 3-9 shows an example ZWaveDevice's property sheet.

Figure 3-9 *Example ZWaveDevice component, property sheet*

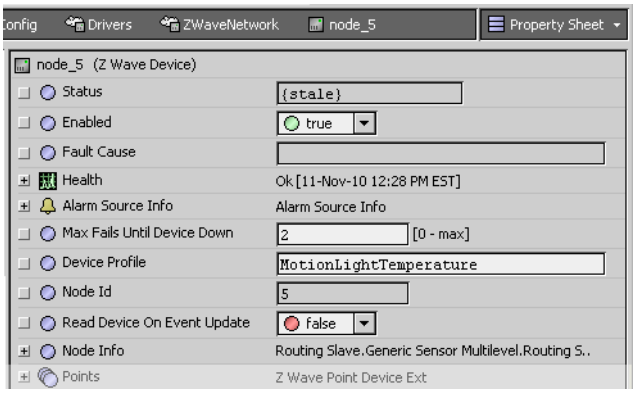


Both Z-Wave device types (ZWaveThermostat, ZWaveDevice) have the same common collection of slots (properties). Depending on the Z-Wave command classes each device type supports, each type is modeled similarly in the NiagaraAX station by child components and proxy points.

ZWaveDevice common properties

Common properties (or slots) on any ZWaveDevice appear near the top of its property sheet, as shown in Figure 3-10.

Figure 3-10 *ZWaveDevice property sheet, showing properties and slots common among all ZWaveDevices*



These common Z-Wave device properties or slots are described as follows:

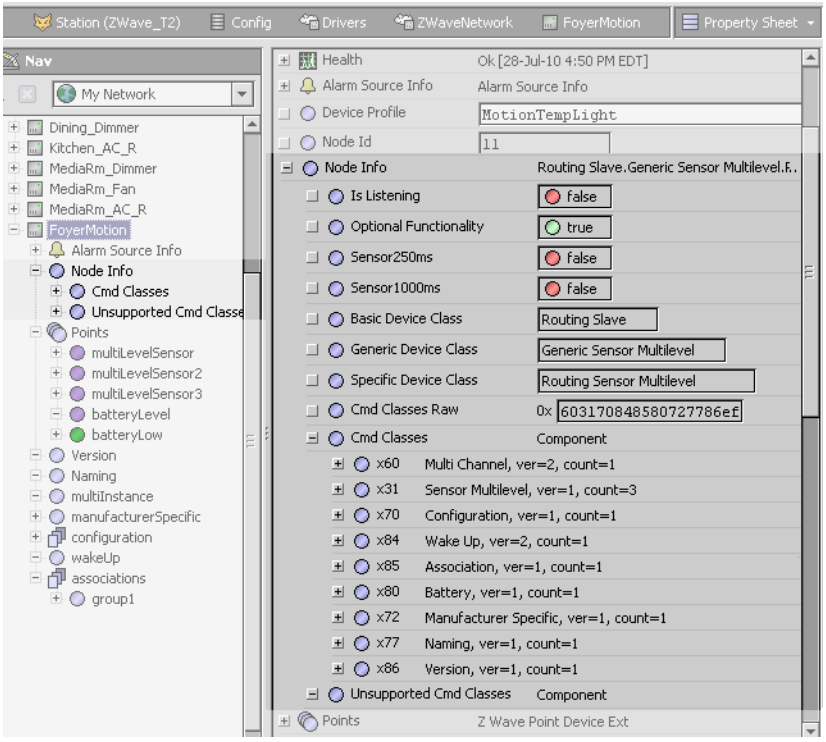
- **Status, Enabled, Fault Cause, Health, Alarm Source Info**
  These properties operate the same as for a device object in most drivers. see "Common device components" in the *Drivers Guide* for general information on these properties.
- **Max Fail Until Device Down**
  Specifies how many unanswered device pings must occur before the driver marks this device with a "down" status, where the default value is 2.
- **Device Profile**
  Used to define and display the "Device Profile" for this specific type of device. See "About Device Profiles" on page 3-32.
- **Node Id**
  (read-only) The unique Z-Wave node address of the device. It is originally assigned by a Z-Wave primary controller when a Z-Wave device is installed / added to a network.

- **Read Device On Event Update**
  This property is *visible only if* this device is *a non-listening (battery-powered) device*, and has a default value of false. It controls whether or not the driver attempts to read all Z-Wave device points and components upon receiving an association event message.
  Note: *If the device is a listening device, it always reads all Z-Wave device points and components on an event message.*
  Some non-listening battery-powered devices stay awake long enough to read this data, and others do not. If you set this to true and the device does not stay awake long enough to read all the data, it could cause several seconds of communications delay due to communications timeouts.
- **Node Info**
  Container slot that lists the Z-Wave device type and supported command classes. See "Node Info".
- **Points**
  A standard Points device extension (container) for data items in this Z-Wave device that need to be polled for data. See "Points".

Node Info

Among ZWaveDevice common properties, **`Node Info`** (Node Information) is a container slot for all node information returned from *any* Z-Wave device. Figure 3-11 shows an example.

Figure 3-11 *Node Info expanded in ZWaveDevice property sheet*



Expand **`Node Info`** to see all its read-only properties, including:

- **Is Listening** — Indicates if this node is always listening or not. Typically this is false for any battery-powered node that sleeps.
  The driver does not attempt communications to a non-listening device that is asleep. If the device also implements the WakeUp command class, it uses the Wake Up Notification message from the device as an indication that the device is awake, and attempts to read all the device's data at that time.
- **Optional Functionality** — Indicates that this node supports other command classes in addition to mandatory classes for its selected generic and specific device class.
- **Sensor250ms** — If true, indicates that this node is an FLiRS (Frequently Listening Routing Slave) device with a wakeup interval of 250ms.
- **Sensor1000ms** — If true, indicates that this node is an FLiRS (Frequently Listening Routing Slave) device with a wakeup interval of 1000ms.
- **Basic Device Class** — Identifies the Z-Wave library used by this device. Possible values include: `Portable Controller`, `Static Controller`, `Slave`, and `Routing Slave`.
- **Generic Device Class** — Identifies the main functionality of this device. For Generic Device Class descriptions, refer to document 2 in "Z-Wave reference documents" on page 1-1.

- **Specific Device Class** — Identifies a specific variant of the Generic Device Class that applies to this device. Again, refer to document 2 in "Z-Wave reference documents" on page 1-1 for descriptions.
- **Command Classes Raw** — A byte array of identifiers for all command classes supported by this device, as received from the device.
- **Cmd Classes** — Container for all Z-Wave command classes supported by the device, where each command class is represented as an expandable node with properties.
  Command classes are listed using the format:

  `xNN  CommandClassType, ver=x, count=y`

  where *NN* is the code for that command class, *x* is version number, and *y* is number of instances.

  For example:

  `x31  Sensor Multilevel, ver=1, count=3`

  Expanding any listed command class simply shows (again) its Cmd Class Type, Version, and Instances (count) as separate properties.
- **Unsupported Cmd Classes** — Container for all Z-Wave command classes supported by the device, but *not supported* by the driver. Unsupported command classes are listed using the format:

  `xNN  CommandClassType`

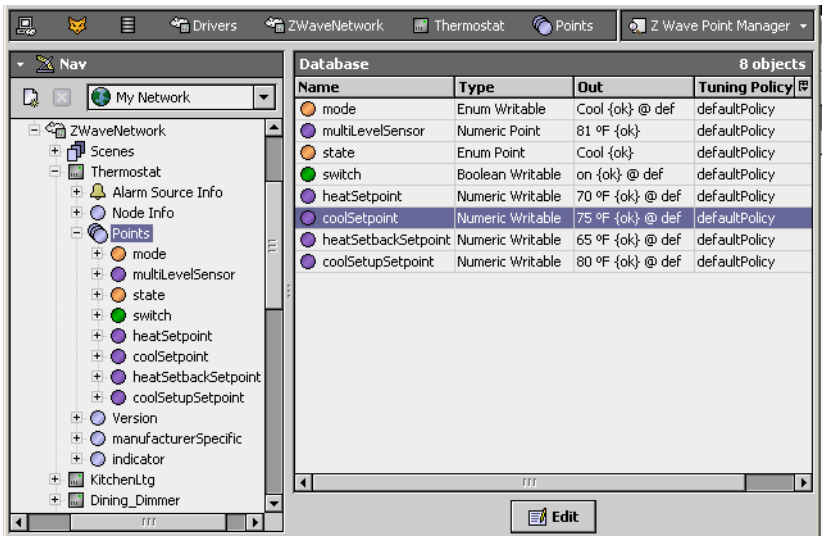
  This container should be empty if all the device's command classes are supported by the driver.

A "Create Command Classes" *action* is also avaiable on NodeInformation (**`Node Info`**). This command automatically occurs upon any device discovery. This creates all the dynamic components under the ZWaveDevice and in its **`Points`** container.

Points

Among ZWaveDevice common properties, **`Points`** (ZWavePointDevExt) is a container slot for all dynamically added control points (Z-Wave proxy points) for that device. Proxy points correspond to any Z-Wave command classes best modeled as control points, for real-time monitoring and control.

Figure 3-12 *Points of a ZWaveDevice, in the default ZWave Point Manager view of the device extension*



Some Z-Wave command classes result in (one or more) dynamically created proxy points, while others result (instead) with dynamically created "child components" directly under the ZWaveDevice parent. See the section "ZWaveDevice child components" for more information.

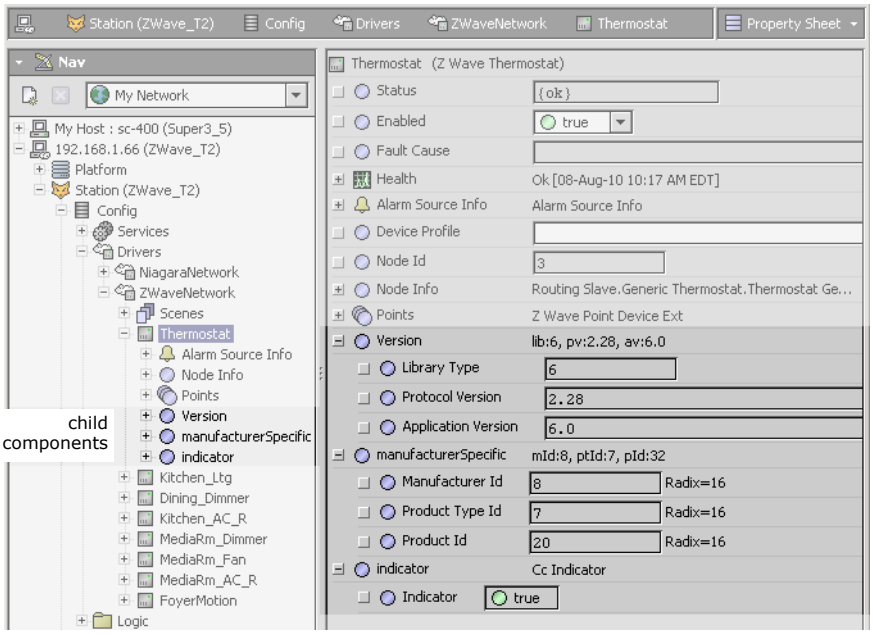
A few Z-Wave command classes result in both ZWaveDevice child components *and* proxy points. For a complete listing of how Z-Wave component modeling occurs for Z-Wave command classes, arranged alphabetically by Z-Wave command class name, see "Z-Wave command classes supported" on page 1-2.

For more details on Z-Wave proxy points, "About ZWave proxy points" on page 3-21.

ZWaveDevice child components

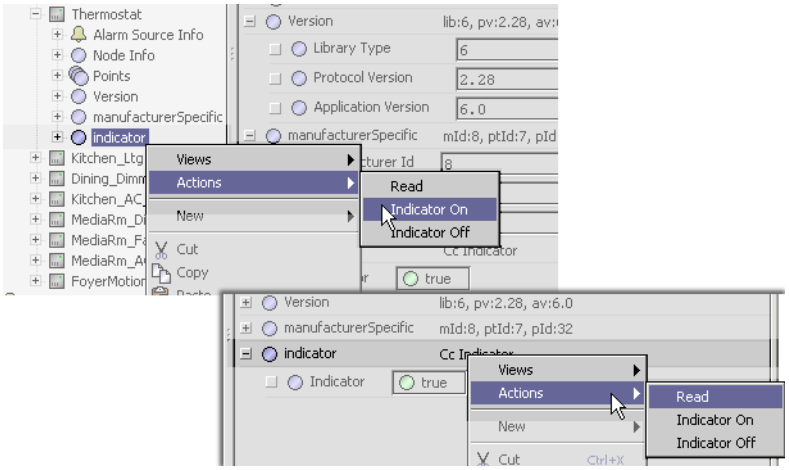
Depending on the Z-Wave command classes supported, a ZWaveDevice has some number of additional dynamic "child components". By default, these components are seen directly *under* the device's **Points** extension when expanding the device in the Nav tree, and also in the device's property sheet (Figure 3-13).

Figure 3-13 *Example dynamic child components under a ZWaveDevice*



Child components have default names such as "**Version**", "**Naming**", "**manufacturerSpecific**", and so on. Often child components have right-click *actions* to read or write data to/from the Z-Wave device. Actions on these components are available in the Nav tree or in the property sheet.

Figure 3-14 *Actions on child components available in Nav tree or ZWaveDevice's property sheet*



In some cases, a few actions are "promoted" to be actions on the parent ZWaveDevice component. For example, two "Assign" actions for the "Naming" child component (Node Name, Node Location) are available on the ZWaveDevice. For related details, see "ZWaveDevice actions".

Child component types for ZWaveDevice objects are listed in the following section.

Child component types

Child components are immediately recognizable by their default *name*, which corresponds to a particular Z-Wave command class. Table 3-1 alphabetically lists current types of child components for ZWaveDevices by default name, and provides a link to a reference topic for each type.

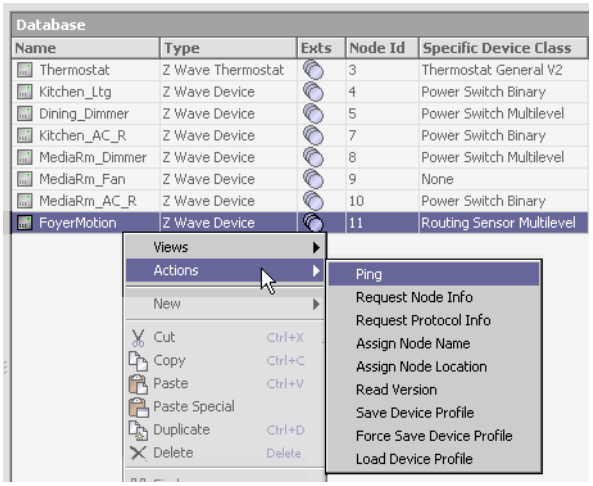
Table 3-1 *Alphabetical listing (by default name) of possible ZWaveDevice child components*

Component under ZWaveDevice (reference information)	Z-Wave Command Class
applicationStatus (CcApplicationStatus on page 5-37)	Application Status
associations (ZWaveAssociationDeviceExt on page 5-42)	Association
associations (ZWaveAssociationDeviceExt on page 5-42)	Multi Channel Association
basicTariffInfo (CcBasicTariffInfo on page 5-38)	Basic Tariff Information
clock (CcClock on page 5-38)	Clock
configuration (ZWaveConfigDeviceExt on page 5-43)	Configuration
hrvControl (CcHrvControl on page 5-38)	HRV Control
hrvStatus (CcHrvStatus on page 5-39)	HRV Status
indicator (CcIndicator on page 5-40)	Indicator
manufacturerSpecific (ZWaveManufacturerSpecific on page 5-43)	Manufacturer Specific
multiInstance (CcMultiInstance on page 5-40)	Multi Instance *or* Multi Channel
Naming (ZWaveNames on page 5-44)	Node Name & Location
protection (CcProtection on page 5-40)	Protection
Scenes (ZWaveSceneNetworkExt on page 5-45)	Scene Activation
switchAll (CcSwitchAll on page 5-41)	All Switch
time (CcTime on page 5-41).	Time
users (ZWaveUserDeviceExt on page 5-45)	User Code
version (ZWaveVersion on page 5-46).	Version
wakeUp (CcWakeUp on page 5-41)	Wake Up

ZWaveDevice actions

A ZWaveDevice has several actions, with an example shown in Figure 3-15 below.

Figure 3-15 *Example actions on ZWaveDevice component*



The total number of actions can vary by which Z-Wave command classes supported by the modeled device. Typical ZWaveDevice actions are described as follows:

- **Ping**
  Forces a ping message to be sent to the Z-Wave device.
- **Request Node Info**
  Forces a "Node Information" request to the device, to retrieve its supported device class and command class data (stored in the ZWaveDevice's "Node Info" slot). See "Node Info" on page 3-16.
- **Request Protocol Info**
  Forces a "Node Protocol Information" request to the device, to retrieve its "Is Listening", "Optional Functionality", "Sensor250ms", and "Sensor1000ms" flags (stored in the ZWaveDevice's "Node Info" slot). See "Node Info" on page 3-16.

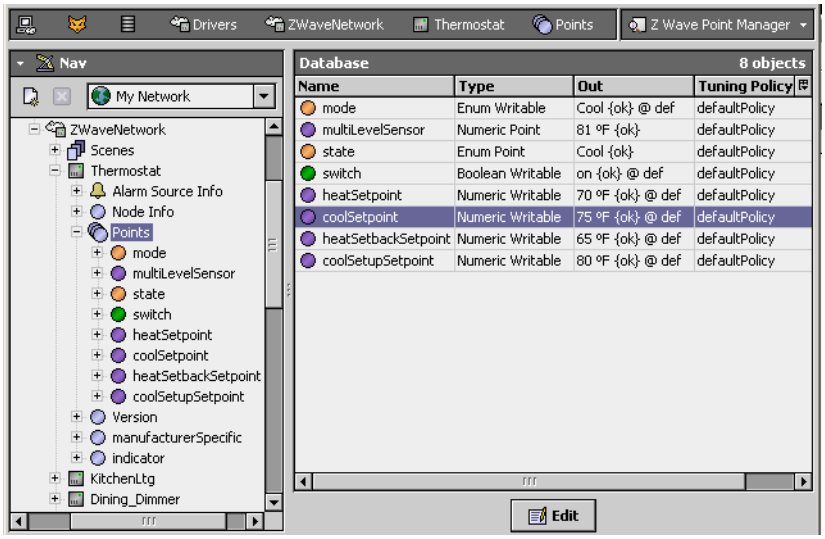
- **Assign Node Name**
  (If device supports "Naming" command class). Produces a popup **`Assign Node Name`** dialog, in which you can type in a text string for node name, which is written to the device. See ZWaveNames on page 5-44 for reference details.
- **Assign Node Location**
  (If device supports "Naming" command class). Produces a popup **`Assign Node Location`** dialog, in which you can type in a text string for node location, which is written to the device.
- **Read Version**
  (If device supports "Version" command class). Forces a "Version" request to the device, with data put in the ZWaveDevice's "Version" slot. See "zwave-ZWaveVersion" on page 5-46 for reference details.
- **Save Device Profile**
  Saves a "Z-Wave device profile" (.info) file for this type of device to the station's file system, providing one does not already exist. See "About Device Profiles" on page 3-32.
- **Force Save Device Profile**
  Forces saving a "Z-Wave device profile" (.info) file for this type of device to the station's file system, overwriting any existing file. See "About Device Profiles" on page 3-32.
- **Load Device Profile**
  Loads any matching "Z-Wave device profile" (.info) file from the station's file system into this device.

ZWave Point Manager

The **`ZWave Point Manager`** (Figure 3-16) is the default view for the "**`Points`**" device extension for any ZWaveDevice, and is similar to other point managers (with differences noted below). See "About the Point Manager" in the *Drivers Guide* for general information.

Note: *The ZWave Point Manager is unique from most Point Managers because there are no "add", "new", or "discover" buttons—all proxy points are added* automatically *(dynamically) as a result of "replication" from the primary controller, or (if station is an SIS or SUC), whenever the parent ZWaveDevice is dynamically added. Only an* **`Edit`** *button is available in this view.*

Figure 3-16 *ZWave Point Manager is default view for Points under a ZWaveDevice*



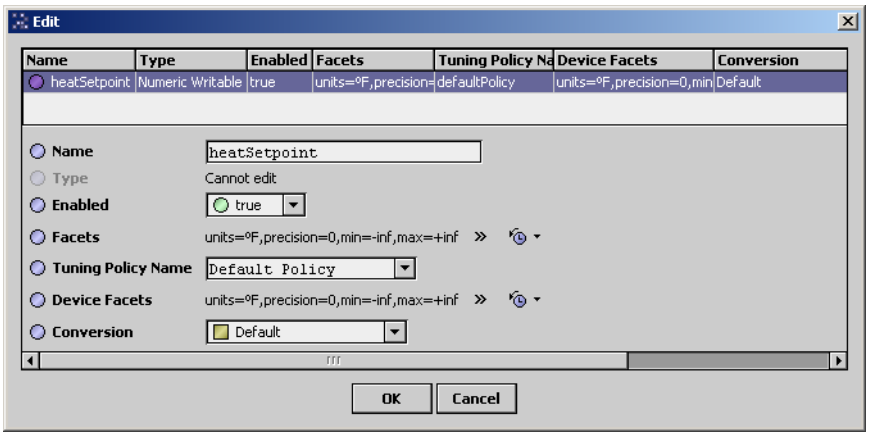
From this view you can review and edit ZWave proxy points. See the next section "Editing ZWave proxy points" for details.

Additional proxy point details are in "About ZWave proxy points" on page 3-21, including a complete listing of possible points, by default name. See "ZWave proxy points by default name" on page 3-23.

Editing ZWave proxy points

Below is an example **Edit** dialog for a ZWave proxy point. To edit a point, double-click it in the ZWave Point Manager, or select one or more points and click the **Edit** button.

Figure 3-17 *Edit dialog for ZWave proxy point*



Change values as needed, using the guidelines below.

- **Name**
  If desired, type in to replace the default Z-Wave proxy point name, where each name must be unique in the parent container for the proxy point.
- **Type**
  Cannot edit (type is ZWaveProxy or ZWaveThermostatSetPointProxy).
- **Enabled**
  Accept the default "true" (if set to "false" point status is set to disabled, and the item is not polled).
- **Facets**
  Edit as needed for proper unit display and decimal handling of the selected item. If a numeric writable, a min or max (other than defaults -inf and +inf) may be desired to limit what can be entered from an override or set action.
  Note: *ZWave proxy points typically have an appropriate unit preselected, based on command class.*
- **Tuning Policy Name**
  Accept the standard "Default Policy", or if additional tuning policies have been added to the ZWaveNetwork, select another tuning policy if needed. If this is point is in a FLiRS device, (Frequently Listening Routing Slave) the tuning policy will set to "flirsPolicy"by default.
- **Device Facets**
  Accept defaults—usage is not applicable in ZWave driver.
- **Conversion**
  Typically, the "Default" conversion is the only useful conversion selection.
  In the case of a Z-Wave proxy point based on a BooleanPoint or BooleanWritable, where you need to "flip" the output logic, you can select the conversion type **Reverse Polarity**.

About ZWave proxy points

ZWave proxy points are similar to other driver's proxy points, as "point-level" components in the NiagaraAX architecture. See "About proxy points" in the *Drivers Guide* for general information.

The following sections provide driver-specific details about ZWave proxy points:

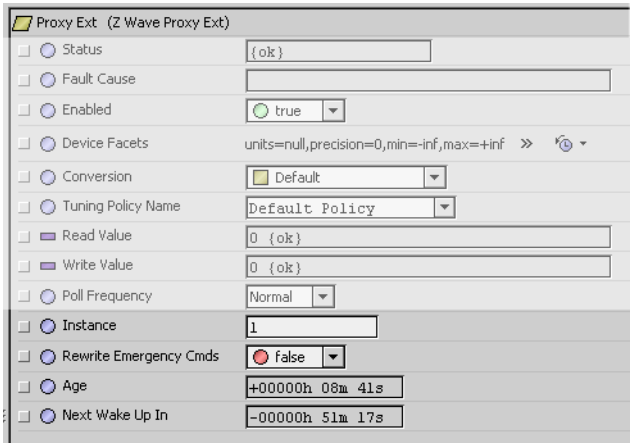
- ZWaveProxyExt properties
- ZWave proxy point actions
- ZWave proxy points by default name

ZWaveProxyExt properties

ZWave proxy points typically use a **ZWaveProxyExt**—except for setpoints under a ZWaveThermostat device, which use a **ZWaveThermostatSetPointProxyExt**. Both proxy extension types share the same set of properties. Each is a proxy for one piece of data of interest in a Z-Wave device. This single piece of data is determined by the corresponding Z-Wave command class, and results in a descriptive default name for the parent Z-Wave proxy point.

Figure 3-18 shows an example ZWaveProxyExt expanded in the property sheet of ZWave proxy point.

Figure 3-18 *ZWaveProxyExt example expanded in property sheet*



In addition to typical ProxyExt properties, the proxy extension for any ZWave proxy point includes these Z-Wave-specific properties:

- **Instance**
  Used for any point, the Instance number from 1 to *n*. Typically this is 1 unless the device's associated Z-Wave command class has a "count" more than 1. For related details, see the section on "Node Info" on page 3-16. Also refer to the target Z-Wave device's vendor documentation for application details.
- **Rewrite Emergency Cmds**
  Used only when the parent point is a writable point, with a default value of false. Setting this to true causes the driver to *rewrite* the Write Value when the following are both true:
  - the Read Value does not match the Write Value, while
  - the active level is Level 1 or Level 2.
- **Age**
  Displays the current age of this value. This value updates only when the point is subscribed and the poll method is executed. Its main purpose is in support of non-listening (battery-powered) devices to provide time information about the data.
- **Next Wake Up In**
  Used only for points in non-listening devices that implement the WakeUp command class, it displays count down time until this value should update from the WakeUp command class. This value only updates when the point is subscribed and the poll method is executed.

ZWave proxy point actions

Unlike proxy points in some other drivers, no "special" actions are available on ZWave proxy points apart from the usual actions on writable types (NumericWritable, BooleanWritable, EnumWritable). A few exceptions are possible for a few ZWave proxy points (Alarm Sensor related ones, Binary Toggle).

If needed, you can go to the slot sheet of these proxy points to edit display names for actions. Or, edit config flags for actions to hide or change access (permission) level.

Note that the *proxy extension* (ZWaveProxyExt or ZWaveThermostatSetPointProxyExt) on any ZWave proxy point has two actions: **`Set On`**, and **`Set Off`**.

Note: *It is recommended that you do not expose or use these proxy extension actions (Set On, Set Off) in any user interface to the system. Unintended control is possible to writable points, with other issues possible.*

ZWave proxy points by default name

ZWave proxy points are recognizable by their default *name*, which corresponds to a particular Z-Wave command class. Table 3-2 alphabetically lists current proxy points for ZWaveDevices by default name.

Table 3-2 *Alphabetical listing (by default name) of possible ZWaveDevice proxy points*

Proxy point default name	Z-Wave "Command Class" and reference link
alarm	"Alarm Sensor" on page 3-24
autoChangeoverSetpoint	"Thermostat Setpoint" on page 3-26
awayHeatSetpoint	"Thermostat Setpoint" on page 3-26
batteryLevel	"Battery" on page 3-24
batterySensor	"Battery Sensor" on page 3-24
bypass	"HRV Control" (CcHrvControl on page 5-38)
co2Alarm	"Alarm Sensor" on page 3-24
coAlarm	"Alarm Sensor" on page 3-24
coolSetpoint	"Thermostat Setpoint" on page 3-26
coolSetupSetpoint	"Thermostat Setpoint" on page 3-26
dischargeTemp	"HRV Status" (CcHrvStatus on page 5-39)
dryAirSetpoint	"Thermostat Setpoint" on page 3-26
e2rateConsumption	"Basic Tariff Information" (CcBasicTariffInfo on page 5-38)
exhaustTemp	"HRV Status" (CcHrvStatus on page 5-39)
fanMode	"Thermostat Fan Mode" on page 3-25
fanState	"Thermostat Fan State" on page 3-25
heatAlarm	"Alarm Sensor" on page 3-24
heatSetbackSetpoint	"Thermostat Setpoint" on page 3-26
heatSetpoint	"Thermostat Setpoint" on page 3-26
hrvControl	"HRV Control" (CcHrvControl on page 5-38)
instantEnergy	"Energy Production" on page 3-24
lock	"Lock" on page 3-25
lowBattery	"Battery" on page 3-24
meterValue	"Meter" on page 3-25
mode	"Thermostat Mode" on page 3-25
mtpWindowCovering	"Move To Position Window Covering" on page 3-25
multilevelSensor	"Multilevel Sensor" on page 3-25
outsideTemp	"HRV Status" (CcHrvStatus on page 5-39)
pulseCount	"Pulse Meter" on page 3-25
rateConsumption	"Basic Tariff Information" (CcBasicTariffInfo on page 5-38)
remainingFilterLife	"HRV Status" (CcHrvStatus on page 5-39)
roomTemp	"HRV Status" (CcHrvStatus on page 5-39)
roomRelHum	"HRV Status" (CcHrvStatus on page 5-39)
sensorBattery	"Battery Sensor" on page 3-24
smokeAlarm	"Alarm Sensor" on page 3-24
state	"Thermostat Operating State" on page 3-26
supplyTemp	"HRV Status" (CcHrvStatus on page 5-39)
switch	"Binary Switch" on page 3-24
switch	"Multilevel Switch" on page 3-25
switchToggle	"Binary Toggle" on page 3-24
todaysEnergy	"Energy Production" on page 3-24
totalEnergy	"Energy Production" on page 3-24
totalTime	"Energy Production" on page 3-24

Proxy point default name	Z-Wave "Command Class" and reference link
ventilationRate	"HRV Control" (CcHrvControl on page 5-38)
waterLeakAlarm	"Alarm Sensor" on page 3-24

Alarm Sensor

A device that supports the `Alarm Sensor` command class supports one or more sensor types. For each supported sensor type, a BooleanPoint is created in the device's **`Points`** container. Table 3-3 lists different Z-Wave alarm sensor types to the default names of corresponding BooleanPoints.

Note: *Alarm extensions are* not *added (by default) to these BooleanPoints. You must add an alarm extension to any point that you want integrated into the station's NiagaraAX alarming subsystem.*

Table 3-3 *Z-Wave sensor type to default name of ZWave proxy BooleanPoint for alarming*

Default Point Name	Z-Wave Sensor Type	Point type, note
alarm	General Purpose Alarm	BooleanPoint
smokeAlarm	Smoke Alarm	
coAlarm	CO Alarm	
co2Alarm	CO2 Alarm	
heatAlarm	Heat Alarm	
waterLeakAlarm	Water Leak Alarm	

Note: *A "silenceAlarm" action is available on any of the points above if the device supports the Alarm Silence command class.*

Alarm Silence

A device that supports the `Alarm Sensor` command class may also support the `Alarm Silence` command class. For each alarm sensor BooleanPoint added by the Alarm Sensor command class, this command class adds a "**`silenceAlarm`**" action. When invoked, it sends a message to the device to silence the device's *local sounding* of an alarm.

Battery

If the `Battery` command class is supported, two points are created in the device's **`Points`** (Table 3-4).

Table 3-4 *Battery default name of ZWave proxy point with description and point type*

Default Point Name	Description	Point type
batteryLevel	Percentage of full battery	NumericPoint
lowBattery	Low battery warning	BooleanPoint

Battery Sensor

If the `Battery Sensor` command class is supported, a BooleanPoint with the default name of "**`sensorBattery`**" is added to **`Points`**. This point indicates if the sensor has detected an event.

Binary Switch

If the `Binary Switch` command class is supported, a BooleanWritable with the default name of "**`switch`**" is added to **`Points`**. This point can be used to both control and indicate the current state of the switch.

Binary Toggle

If the `Binary Toggle` command class is supported, a BooleanPoint with the default name of "**`switchToggle`**" is added to **`Points`**. A dynamic *action* named "Toggle" is added to this point. This proxy point indicates the current state of the switch, and its action is used to command the device to toggle the current state of the switch.

Energy Production

A device that supports the `Energy Production` command class retrieves energy production data from an energy production device (e.g. generator). If the device supports this command class, four NumericPoints are added to **`Points`**, as described in Table 3-5.

Table 3-5 *Energy Production default ZWave proxy point names to parameters, units, point types*

Default Point Name	Production Parameter	Units	Point type
instantEnergy	Instant energy production	Watts	NumericPoint
totalEnergy	Total energy production	Watt-Hours	
todaysEnergy	Energy production today	Watt-Hours	
totalTime	Total production time	Hours or Seconds	

Hail

A device that supports the `Hail` command class fires the "**`hail`**" topic of its corresponding ZWaveDevice component whenever a Z-Wave "Hail" message is received from another device. Usage of the Hail command class is application-specific.

Lock

If the `Lock` command class is supported, a BooleanWritable with the default name of "**`lock`**" is added to **`Points`**. This point can be used to both control and indicate the current lock state of a lockable device.

Meter

If the `Meter` command class is supported, a NumericPoint with the default name of "**`meterValue`**" is added to **`Points`**. This point reads the accumulated values from an electric, gas, or water meter. The point's units are automatically set to match the units reported by the Z-Wave device.

Move To Position Window Covering

If the `Move To Position Window Covering` command class is supported, a NumericWritable with the default name of "**`mtpWindowCovering`**" is added to **`Points`**. This point can indicate the current position of the window covering, and can move the window covering to a given position.

Multilevel Sensor

If the `Multilevel Sensor` command class is supported, a NumericPoint with the default name of "**`multiLevelSensor`**" is added to **`Points`**. This point indicates the current value of the multilevel sensor, in the units reported by the Z-Wave device.

Multilevel Switch

If the `Multilevel Switch` command class is supported, a NumericWritable with the default name of "**`switch`**" is added to **`Points`**. This point indicates the current value of the multilevel switch, in the units reported by the Z-Wave device, and can be commanded to change the value.

Pulse Meter

If the `Pulse Meter` command class is supported, a NumericPoint with the default name of "**`pulseCount`**" is added to **`Points`**. This point indicates the current accumulated pulse counts reported by a pulse counting utility meter device.

Thermostat Fan Mode

If the `Thermostat Fan Mode` command class is supported, an EnumWritable with the default name of "**`fanMode`**" is added to **`Points`**. This point monitors, and can be used to control, the current fan mode of the HVAC system.

The driver queries the Z-Wave device to determine the fan modes supported. If the fan is single-speed, mode choices are "Auto" and "On". If a two-speed fan, mode choices are "auto", "onLow", "autoHigh", and "onHigh".

Thermostat Fan State

If the `Thermostat Fan State` command class is supported, an EnumPoint with the default name of "**`fanState`**" is added to **`Points`**. This point monitors the current fan state of the HVAC system. Valid fan states include "idle", "on", and "onHigh".

Thermostat Mode

If the `Thermostat Mode` command class is supported, an EnumWritable with the default name of "**`mode`**" is added to **`Points`**. This point monitors, and can be used to control, the thermostat operation mode of the HVAC system.

The driver queries the Z-Wave device to determine the thermostat modes supported. Supported modes are added as enumerations to the EnumWritable, with possible enum tags shown in Table 3-6.

Table 3-6 *Possible Enum Tag values and descriptions for Thermostat "mode" ZWave proxy point*

Enum Tag	Description
off	Off — System is off. No heating or cooling will come on.
heat	Heat — Only heating will occur.
cool	Cool — Only cooling will occur.
auto	Auto — Heating or cooling will come on according, to the heating and cooling setpoints. The system automatically switches between heating and cooling when the temperature exceeds (or drops below) the setpoints.
aux	Auxiliary/Emergency Heat — A heat pump, especially an exchange type, is not efficient when outside temperature is below 35 degrees Fahrenheit (close to 0 degrees Celsius). Thus, the thermostat may be put into an "Aux heat mode" simply to use a more efficient secondary heat source, when there are no failures with the heat pump or its compressor.
resume	Resume — The system resumes from last active mode.
fan	Fan Only — Only cycle fan to circulate air.
furnace	Furnace — Only cycle fan to circulate air.
dry	Dry Air — The system cycles cooling in relation to the room and setpoint temperatures, in order to remove ambient moisture. (Dehumidification)
moist	Moist Air — Humidification.
autochange	Auto Changeover — Heating or cooling comes on according to the auto changeover setpoint.
heatEnergySave	Energy Save Heat — Energy Save Mode Heating will occur (usually lower than normal setpoint).
coolEnergySave	Energy Save Cool — Energy Save Mode Cooling will occur (usually higher than normal setpoint).
away	AWAY — Special heating mode, to prevent water from freezing in forced water systems.

Thermostat Operating State

If the `Thermostat Operating State` command class is supported, an EnumPoint with the default name of "**`state`**" is added to **`Points`**. This point monitors the HVAC system's operating state.

Possible HVAC system operating states are shown in Table 3-7.

Table 3-7 *Possible operating states for Thermostat "state" ZWave proxy point*

Enum Tag	Description
idle	System is idle.
heat	Heating.
cool	Cooling.
fan	Fan Only.
heatPend	Short cycle prevention feature used in heat pump applications to protect the compressor.
coolPend	Short cycle prevention feature used in heat pump applications to protect the compressor.
vend	Vent / Economizer.

Thermostat Setpoint

If the `Thermostat Setpoint` command class is supported, one or more NumericWritable points are added to **`Points`**, for setpoint handling. These points are used to monitor and control the setpoints supported by the device. Table 3-8 provides a list of possible proxy points that may be added.

Table 3-8 *Default ZWave proxy point names associated with Thermostat Setpoints*

Default Point Name	Setpoint	Point type
heatSetpoint	Heating	NumericWritable
coolSetpoint	Cooling	
furnaceSetpoint	Furnace	
dryAirSetpoint	Dry Air	
moistAirSetpoint	Moist Air	
autoChangeoverSetpoint	Auto Changeover	
heatSetbackSetpoint	Energy Save Heating	
coolSetupSetpoint	Energy Save Cooling	
awayHeatSetpoint	Away Heating	

Typically, a device does not have all 9 setpoints.

About Z-Wave Scenes

Scenes are a pre-set combination of actions that occur at the touch of a single button. A scene can be any combination of Z-Wave device control settings. For example, providing that the associated Z-Wave devices support scenes, you can create a scene for sunset that closes the blinds, turns on inside lights to a pre-set level, and activates an outdoor motion sensor. However, scene support is most common amongst Z-Wave lighting control devices.

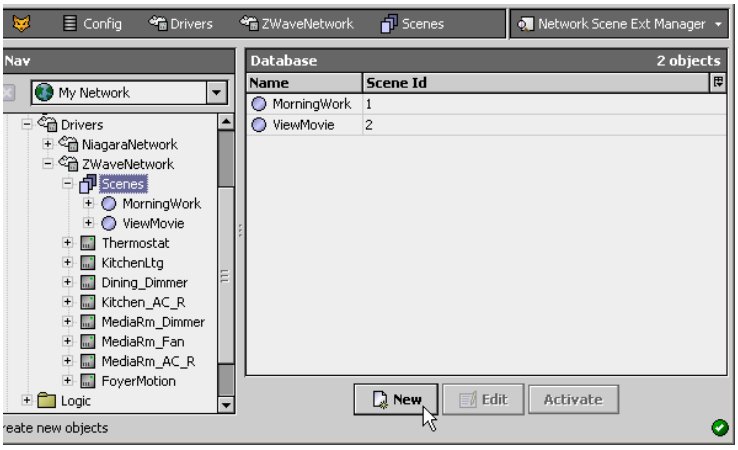
Creating and setting up scenes

In the NiagaraAX station, scenes are modeled at the network level using a "**Scenes**" network extension on the **ZWaveNetwork**. A special Workbench view on **Scenes**, as well as on each of the "Network-Scene" (scene) children, provide an interface to quickly define a scene.

To create a scene

Step 1 Double-click the **Scenes** network extension under the **ZWaveNetwork** to access the **Network Scenes Ext Manager** (Figure 3-19). Or access this view using the view selector for **Scenes**.

Figure 3-19 *Network Scene Ext Manager*

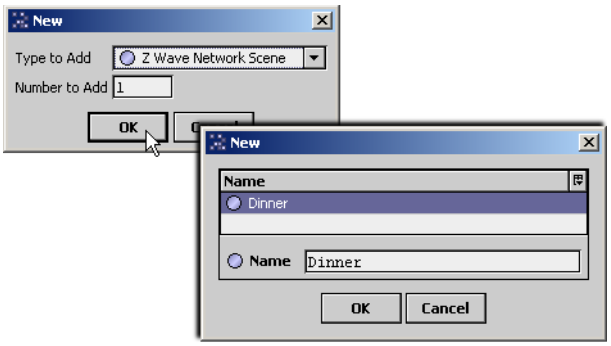


The **Network Scene Ext Manager** lists any existing scenes, if applicable.

Step 2 Click the **New** button in this manager view.

A **New** popup dialog opens, for type "Z Wave Network Scene", with "Number to Add" (Figure 3-20). Type in the number to add, or just click **OK** to add one.

Figure 3-20 *Popup dialogs when adding new scene in Network Scene Ext Manager*



In the second **New** dialog, type in a name for each scene, replacing "ZWaveNetworkScene", and click **OK**. In this example, the new scene is named "Dinner".

This new scene (ZWaveNetworkScene component) is now added, but not yet defined.

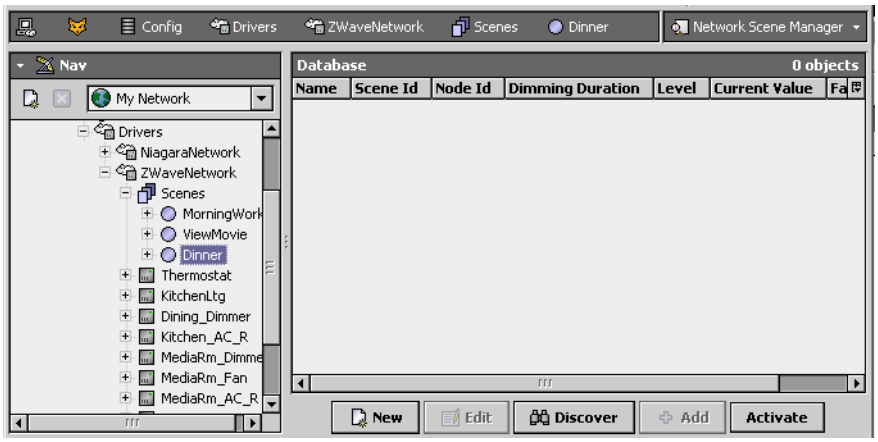
Note: *A numerical Z-Wave "scene ID" is automatically assigned, but not yet visible unless you go to its property sheet. Scene IDs must be unique integers, starting with 1.*

To setup a scene

Step 1 Double-click the scene added (for example, "Dinner"), in either the **Network Scenes Ext Manager**, or in the Nav tree, with the parent **Scenes** extension expanded.

The view *changes* to the **Network Scene Manager** for that scene (Figure 3-21).

Figure 3-21 *Network Scene Ext Manager*

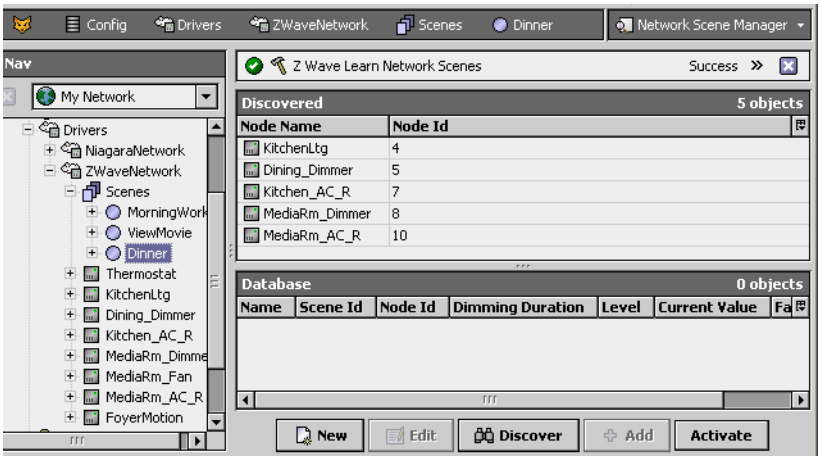


For a newly-added scene, the Database pane will be *empty*, as shown above.

Step 2 Click **Discover**.

The view splits into two-pane Learn mode: **Discovered** top and **Database** lower. A discovery job launches to find devices that support scenes—and if this scene ID is already specified in each (Figure 3-22).

Figure 3-22 *Following scene Discover in the Network Scene Manager view, showing available devices*

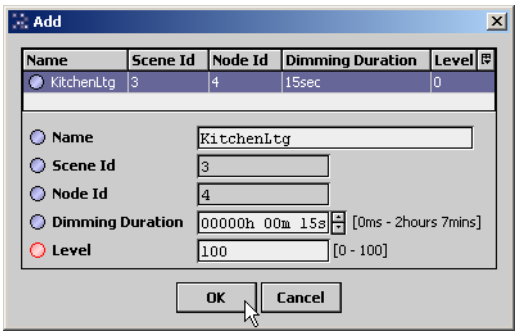


In the Figure 3-22 example above, following a discovery, scene "Dinner" shows 5 available nodes that support scenes. None are yet specified in this scene.

Step 3 In the Discovered pane, double-click a node to add it. Or, hold Ctrl while clicking multiple nodes and then click **Add**.

A popup **Add** dialog for the node (or nodes) appears, as shown in Figure 3-23.

Figure 3-23 *Add dialog for specifying node in a Z-Wave scene*

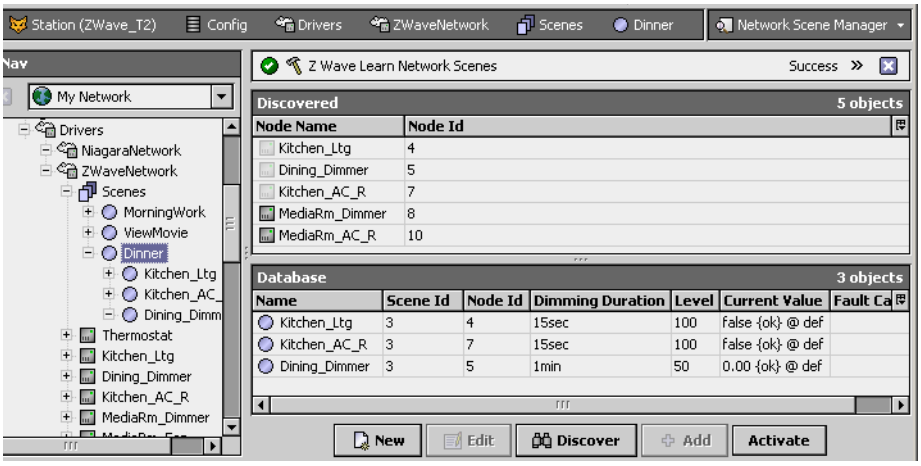


Step 4 For each node in the **Add** dialog, enter:

- **Dimming Duration**
  Applies if a dimmer node only. Specifies the "ramp time" from the current dim level to the final specified Level for the scene. (Ignored if a non-dimmer node—the Level is applied immediately).
- **Level**
  Enter any value from 0 to 100.
  - If a dimmer node, this specifies the final dim level at the end of the Dimming Duration.
  - If a binary node, a "0" value means *Off*, any non-zero value (1 - 100) means *On*.

And click **OK** to enter in the system. Figure 3-23 shows a scene that includes 3 nodes.

Figure 3-24 *Network Scene Manager view of scene that affects 3 nodes*



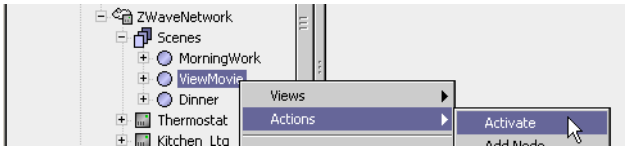
The scene can now be activated, or if necessary, edited again to modify. See "Activating scenes".

Activating scenes

Scenes can be activated a number of different ways:

- While in the **`Network Scene Manager`** view of any scene, click the **`Activate`** button.
- While in the **`Network Scene Ext Manager`** view (of the **`Scenes`** extension), click to *select* one or more scenes, then click the **`Activate`** button.
- In the property sheet of the **`ZWaveNetwork`** expand **`Scenes`**, then right-click any child scene (ZWaveNetworkScene component), and select **`Actions > Activate`**.
- In the Nav tree with **`Scenes`** expanded, right-click any child scene (ZWaveNetworkScene component), and select **`Actions > Activate`**, as shown in Figure 3-25.

Figure 3-25 *Right-click scene in Nav tree or in property sheet to activate*



Also see "About Z-Wave Scenes" on page 3-27.

About Z-Wave Associations

Z-Wave *associations* are application-level connections between Z-Wave nodes. Devices that support associations implement the `Association` command class. In the NiagaraAX station, associations between devices are specified using ZWaveAssociationGroup (**`groupn`**) component children of the ZWaveAssociationDeviceExt (**`associations`**) "child component" of ZWaveDevice components.

However, the easiest way to create and view Z-Wave associations is via the *wire sheet* of view of the **`ZWaveNetwork`**. Here, associations appear as "green" colored links between devices on the wire sheet.

Association types, along with wire sheet examples, are provided in the following sections:

- Single Instance Associations
- Multi Instance Associations
- Multi Channel Associations

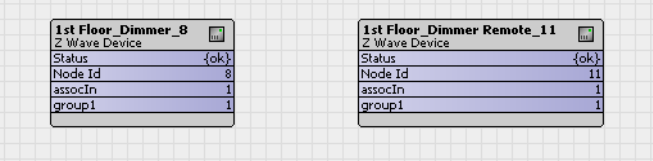
Single Instance Associations

Any device that supports Z-Wave associations has an integer "assocIn" slot exposed on its corresponding ZWaveDevice component. The "assocIn" slot is the *target* of an association link.

For each association group a device supports, it has an integer slot "group*n*", where *n* is the group number. Any "group*n*" slot is a *source* of an association link.

A ZWaveDevice's component glyph (shape) shows "assocIn" and "group*n*" slots, as shown in Figure 3-26.

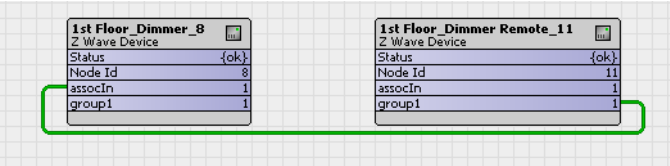
Figure 3-26 *ZWaveDevice shapes on wire sheet of ZWaveNetwork, showing "assocIn" and "groupn" slots*



In Figure 3-26, there is dimmer device (node 8, left) and a dimmer remote device (node 11, right). A dimmer remote is similar to a dimmer device, except that it does not support a direct connection to a lighting load.

In this application, the need is for the dimmer remote node (node 11) to be able to control the light that is wired to the "1stFloor_Dimmer_8" (node 8) device. To provide this, you simply link from node 11's "group1" (source) slot to the node8's "assocIn" (target) slot, as shown below in Figure 3-27.

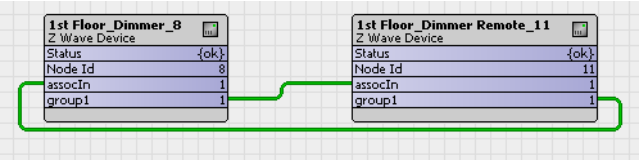
Figure 3-27 *Association made by linking from groupn source to assocIn target*



The link in Figure 3-27 causes an association to be made from node 11 to node 8. When node 11's dimmer switch is activated locally, a Z-Wave message is sent directly to node 8, causing the light wired to node 8 to change to the level set by node 11.

Note if the light is controlled directly at node 8, the dim level displayed at node 11 does not change. However, it is preferred that the current status of the light connected to node 8 to also be indicated at node 11. This requires *adding another link* from node8's "group1" slot to node 11's "assocIn" slot (Figure 3-28).

Figure 3-28 *Association links made in both directions between two nodes*



Multi Instance Associations

If a device supports Multi Instance associations, it is similar to the single instance association. However, instead of the single "assocIn" slot, the device also has multiple "assocIn_*n*" slots, where *n* is the instance number. Each of these "assocIn_*n*" slots are possible link targets.

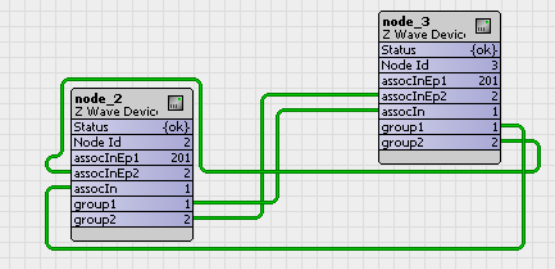
See "Single Instance Associations" on page 3-30.

Multi Channel Associations

If a device supports Multi Channel associations, the association is created to a "end point" in a multi channel device. This is similar to the single instance association, only instead of a single "assocIn" slot, the device also has multiple "assocInEp*n*" slots, where *n* is the end point number.

Figure 3-29 shows an example of Multi Channel association links.

Figure 3-29 *Example Multi Channel association links*



The Figure 3-29 example is from an application that has two nodes, node_2 (left) and node_3 (right). Each node is a 2-gang light switch, that is, each node can control two separate lights. In this case, lights are only wired to node_2.

The need is for switch 1 on either node to control light 1, and for switch 2 on either node to control light 2.

Note: *In the Figure 3-29 example, devices used were Astral LSM12s devices. When the link to associate switch 1 to light 1 was originally added, it was added as a link from "group1" to "assocInEp1".*

However, the device reported back the link as a single instance link to the "assocIn" slot—and as such, the link automatically "moved itself" to that destination slot!

At the time of this document, it is not known if this is "typical behavior" for end point 1 associations on multi channel devices.

About Device Profiles

Device profiles are used to map generic names for Z-Wave proxy points and configuration properties to specific names for a specific Z-Wave device. A device profile can also be used to meet a specific device "contract" or "profile". You can use Workbench as a tool to create these device profiles.

The device profile is an ASCII text file that is named using the information from the manufacturerId, productTypeId, and productId properties from the Z-Wave device. The file contains the device profile name, and a "pointNameMap". The pointNameMap is the definition of a Baja NameMap for the ZWaveDevice's points device extension, providing application-specific display names for specific points.

A simple example is for a Z-Wave device that is used to control a window covering. Such a device typically implements a Z-Wave `Multilevel Switch` command class, used to control the position of the window covering. However, there may not be anything in the Z-Wave device that will indicate that it is a window covering application.

Without any device profile, when the device is discovered, a NumericWritable point is created for the Multilevel Switch command class. This point is named "switch".

Using Workbench, from the slot sheet of the device's **`Points`** extension, you can set the display name for the "switch" point to "percentOpen". You could also set the ZWaveDevice's "Device Profile" property value to "WindowCovering" (note the "Device Profile" property is near the top of the property sheet for any ZWaveDevice component).

After doing this, if you invoked the "Save Device Profile" action on this ZWaveDevice, a file named "`x65_x1_x2.info`" would be generated that contains the following:

```
deviceProfile=WindowCovering
pointsNameMap{switch=percentOpen;}
```

Note: *Device profile files are saved in the station's file space in a `!Files/zwaveDevTemplates` directory.*

Later, during a discovery process, as ZWaveDevices are added to the station, the driver first looks for the existence of a device profile file for a device in `!Files/zwaveDevTemplates`. If one is not found there, the driver looks in the `zwave.jar` (driver) module, as a collection of device profiles are included in the driver. If a device profile file is found, it initializes the "Device Profile" property of the ZWaveDevice component, and sets display names in the "pointNameMap" of its **`Points`** (ZWavePointDeviceExt).

Chapter 4

ZWave Plugin Guides

Plugins provide *views* of components, and can be accessed many ways—for example, double-click a component in the tree for its *default* view. In addition, you can right-click a component, and select from its **Views** menu. For summary documentation on any view, select **Help > On View** (F1) from the Workbench menu, or press F1 while the view is open.

Summary information is provided here about the different ZWave views.

ZWave Plugin Guides Summary

Summary information is provided on views specific to components in the `zwave` module, with views listed in alphabetical order as follows:

- Network Scene Ext Manager
- Network Scene Manager
- ZWave Device Manager
- ZWave Point Manager

zwave-NetworkSceneExtManager

The **Network Scene Ext Manager** is the default view of the **Scenes** extension under the ZWaveNetwork (ZWaveSceneNetworkExt). Use this view to add Z-Wave *scenes* (ZWaveNetwork-Scenes).

Note: *To define a scene, after adding it (New),* double-click it *for its own Network Scene Manager view. For detailed information, see "Creating and setting up scenes" on page 3-27.*

Scenes appear listed in **Network Scene Ext Manager** view with the following default data columns:

- **Name**
  Niagara name for the scene, as entered when adding a New scene or editing an existing scene.
- **Scene Id**
  Z-Wave scene ID number (1 - 255), unique for each scene.

In addition, using the table options control, the following additional data columns are available:

- **Path**
  Niagara station path of each scene component (ZWaveNetworkScene), relative to the root.

zwave-NetworkSceneManager

The **Network Scene Manager** is the default view of any child *scene* under the **Scenes** extension of the ZWaveNetwork. Scenes are defined combinations of actions across one or more devices that can be "activated" by a single command. Use this view to add new nodes for a scene, edit any node in the scene, and activate that scene.

Each entry (row) is a "NetworkSceneConfig" component (ZWaveNetworkSceneConfig) that specifies a particular device (node) and action. For detailed information, see "Creating and setting up scenes" on page 3-27.

By default, the **Network Scene Manager** has following data columns enabled:

- **Name**
  Name of the "ZWaveNetworkSceneConfig" component.
- **Scene Id**

Z-Wave scene ID, integer 1 - 255 (should be the same for all NetworkSceneConfig components)

- **Node Id**
  Z-Wave node ID for that device.
- **Dimming Duration**
  Specified time it takes for the device to ramp to the specified Level.
- **Level**
  Final dimming level, from 0 to 99 (or if a dimmer), or if a Boolean (binary) type, 0 for Off or 255 (or any non-zero value) for On.
- **Current Value**
  Current real-time value of the device.
- **Fault Cause**
  Typically blank, unless a ZWaveNetworkSceneConfig component has a fault status.

In addition, using the table options control, the following additional data column is available:

- **Path**
  Niagara station path of each "NetworkSceneConfig" component (ZWaveNetworkSceneConfig), relative to the root.

zwave-ZWave Device Manager

The **`ZWave Device Manager`** is the default view of a ZWaveNetwork. Use the ZWave Device Manager to review and access Z-Wave device components (ZWaveDevice and ZWaveThermostat) in the station's Z-Wave network.

Although this view features online "discovery" of devices, you typically use the "ReplicateReceive" function to initially configure the network, working with the required Z-Wave "primary controller". Then, depending on whether the JACE will operate as an SUC or SIS (typical), or as a static secondary controller, devices automatically populate the network as they are added one at a time, or all at once.

A number of buttons appear at the bottom of this view. For specific details, see "ZWave Device Manager" on page 3-12.

Added ZWaveDevices appear in the view's table. By default, the following columns appear in this view:

- **Name**
  - If the ZWaveNetwork's property "Use Node Names" is false (default), it shows the name of the device-level component that represents the Z-Wave device.
  - If the ZWaveNetwork's property "Use Node Names" is true, instead of component name you see a text string of "*`Node Location_Node Name_NodeId`*" for each device that supports the the Node Name and Location command class. For example, "`1st Floor_MotionFoyer_11`". Any device not supporting this command class shows its "`specificDeviceClass_NodeId`", for example "`thermostatGeneralV2_3`" instead of component name.
- **Type**
  Component type, currently either "Z Wave Thermostat" or "Z Wave Device".
- **Exts**
  Shows an icon for the **`Points`** de vice extension of a ZWaveDevice—double-click the icon for the ZWave Points Manager.
- **Node Id**
  Unique integer Z-Wave node ID (2 - 255) for the device.
- **Specific Device Class**
  The "Specific Device Class" property of the device's "Node Info"(NodeInformation) component.
- **Device Profile**
  Displays the "Device Profile" for the device, if available. "About Device Profiles" on page 3-32.
- **Node Name**
  The device's "Node Name", from the its "Naming" (ZWaveNames) component (if supported).
- **Node Location**
  The device's "Node Location", from the its "Naming" (ZWaveNames) component (if supported).
- **Health**
  The device's current "health" and timestamp of last update.

In addition, using the table options control, the following additional data columns are available:

- **Status**
  Current status of the ZWaveDevice component.
- **Enabled**
  Whether the ZWaveDevice component is enabled (true) or disabled (false).

zwave-ZWave Point Manager

Use the ZWave Point Manager to review and edit ZWave proxy points under the Points extension of a selected Z-Wave device (ZWaveDevice or ZWaveThermostat). The ZWave Point Manager is the default view on both these components. To view, *double-click* the Points extension, or right-click and select **`Views > ZWave Point Manager`**.

For more details, see "ZWave Point Manager" on page 3-20.

By default, the following columns appear in this point manager:

- **Name**
  Niagara name of the ZWave proxy point. Default names depend on the associated Z-Wave command class.
- **Type**
  Niagara type of NiagaraAX control point used as the proxy point (e.g. NumericPoint, BooleanPoint, EnumPoint, NumericWritable, EnumWritable, or BooleanWritable).
- **Out**
  Current last polled value (out slot) of the proxy point, reflecting status and facets.
- **Tuning Policy Name**
  Assigned tuning policy (Default Policy, or other if available and assigned).

In addition, using the table options control, the following additional data columns are available:

- **Path**
  Station path of the proxy point, relative to the root.
- **Enabled**
  Whether enabled (true, default) or disabled (false).
- **Facets**
  Assigned units (AI or AO), trueText and falseText (BI or BO), or enum range (Enum).
- **Device Facets**
  Typically reflect point facets, if not blank..
- **Conversion**
  Niagara conversion type used, such as Default, Linear, or Reverse Polarity.
- **Age**
  Time since last value update from the Z-Wave device.
- **Next Wake Up In**
  If the parent device is battery-powered, this shows the timestamp of its next anticipated "wake up".
- **Fault Cause**
  If the point is in fault, explains why.
- **Read Value**
  Last read value from device.
- **Write Value**
  Last value written to device (if applicable).

Chapter 5

Z-Wave Component Guides

These component guides provides summary help on Z-Wave components.

Z-Wave Component Guides Summary

Summary information is provided on components specific to the `zwave` module, listed in alphabetical order as follows:

- CcApplicationStatus
- CcBasicTariffInfo
- CcClock
- CcHrvControl
- CcHrvStatus
- CcIndicator
- CcMultiInstance
- CcProtection
- CcSwitchAll
- CcTime
- CcWakeUp
- NodeInformation
- ZWaveAssociationDeviceExt
- ZWaveAssociationGroup
- ZWaveConfigDeviceExt
- ZWaveDevice
- ZWaveManufacturerSpecific
- ZWaveNames
- ZWaveNetwork
- ZWaveNetworkScene
- ZWaveNetworkSceneConfig
- ZWavePointDeviceExt
- ZwaveProxyExt
- ZWaveSceneActuatorConf
- ZWaveSceneDeviceExt
- ZWaveSceneNetworkExt
- ZWaveThermostat
- ZwaveThermostatSetPointProxyExt
- ZWaveUserDeviceExt
- ZWaveVersion

zwave-CcApplicationStatus

CcApplicationStatus (**`applicationStatus`**) is a child container of a ZWaveDevice that supports the `Application Status` command class. It may be used to see if the device is busy or unable to process any application requrest, via read-only child properties including the following:

- **Status** — Indicates the application status last set by the device. Possible values include:
  - Try Again Later
  - Try Again Wait Seconds
  - Request Queued
  - Request Rejected
  - None

- **Wait Time** — If status is "Try Again Wait Seconds", this indicates the wait time.
- **Timestamp** — Timestamp for the last application status message received from this device.

zwave-CcBasicTariffInfo

CcBasicTariffInfo (**basicTariffInfo**) is a child container of a ZWaveDevice object that supports the Basic Tarriff Information command class. It is for use with a single-element or dual-element meter when used with import rates (electricity received from grid) only.

Its read-only properties reflect the meter configuration and its current rate and consumption values.

- **Is Dual** — If true indicates a dual-element meter, otherwise (false) is a single-element meter.
- **Total Number Import Rates** — The number of import rates E1 (and E2) supported by the meter. Value range is 1 - 8.
- **Element1 Current Rate In Use**— Which import rate is currently in use.
- **Element1 Rate Consumption** — Consumption rate for element 1, with units in W-hr.
- **Time Next Rate** — Indicates the time (hh:mm:sec) that the rate is due to change on element 1. It may also indicate "not used".
- **Element2 Current Rate In Use** — Which import rate is currently in use.
- **Element2 Rate Consumption** — Consumption rate for element 2, with units in W-hr.

In addition, one or two Proxy points are added.

Proxy points

If the device supports the Basic Tariff Information command class, one or two proxy points are added to its ZWaveDevice's **Points** container, with default names as follows:

- **rateConsumption**
  NumericPoint that provides energy consumption in Watt-hours (W-hr).
- **rateConsumption2**
  (Only if a dual-element meter) NumericPoint for energy consumption in Watt-hours (W-hr).

zwave-CcClock

CcClock (**clock**) is a child container of a ZWaveDevice object that supports the Clock command class. It can be used for simple clock functionality to display time via read-only child properties, including the following:

- **Unused** — Indicates if the clock is not being used by the device (If true, clock not being used).
- **Time** — If used (Unused = false), this is the current time of day in the device, to minute resolution.
- **Week Day** — If used (Unused = false), this is the current day of the week in the device.

In addition, the following right-click *actions* are on the CcClock (**clock**) component:

- **Read Time** — Forces a time read request to be sent to the device.
- **Sync Time** — Sends a message to the device to set its time and date to match the current time and date in the JACE.

zwave-CcHrvControl

CcHrvControl (**hrvControl**) is a child container of a ZWaveDevice object that supports the HRV Control command class, to control the operation of a HRV (Heat Recovery Ventilation) system. During a discovery, the Z-Wave driver queries the device to determine the control modes and bypass modes that the device supports. and adds one or more Proxy points as a result.

In addition, a right-click action, **Read Modes Supported**, is on the **CcHrvControl** component. This allows you to force the reading of the HRV Control modes supported by the device, where corresponding proxy points are added.

Note: *This is automatically done during the discovery process*

Proxy points

If the device supports the HRV Control command class, three proxy points are added to its ZWaveDevice's **Points** container. Point types and their default names are shown in Table 5-1.

Table 5-1 *HRV Control proxy point default names to HRV control function and point type*

Default proxy point name	HRV Control function	Point type
"hrvControl"	Operation mode	EnumWritable
"ventilationRate"	Set Ventilation Rate	NumericPoint
"bypass"	Bypass Control	NumericPoint

Enum choices for hrvControl Based on the Manual Control Modes supported, the enum choices for the "hrvControl" point include only the values supported by the device, among those in Table 5-2.

Table 5-2 *Descriptions for possible Enum tags for EnumWritable (hrvControl) proxy point*

"hrvControl" Enum Tag	Description
"off"	System is in the off state; frost protection can occur.
"demandAuto"	System is controlled based on sensor input.
"schedule"	System is controlled based on predefined input from the factory and/or the user or installer.
"energySaving"	System is put into a reduced heat / ventilation mode.
"manual"	System is in manual mode.

Action choices for NumericPoints Based on the Manual Control Modes supported, one or more actions are added to the (NumericPoint) "bypass" proxy point, among those in Table 5-3.

Note: *If the "hrvControl" EnumWritable is not in "manual", the device will ignore these manual commands.*

Table 5-3 *Possible actions and descriptions for NumericPoint "bypass" proxy point*

"bypass" Action	Description
"openClose"	Can force the bypass open of closed.
"auto"	Sets the bypass into automatic mode.
"set"	Sets the bypass to a value from 1 to 99.

In addition, the following action may be added to the (NumericPoint) "ventilationRate" proxy point:

- "set" — Sets the manual ventilation rate to a value from 0 to 99.

zwave-CcHrvStatus

CcHrvStatus (**`hrvStatus`**) is a child container of a ZWaveDevice object that supports the `HRV Status` command class, to read a number of HRV (Heat Recovery Ventilation) parameters. During a discovery, the Z-Wave driver queries the device and adds one or more Proxy points as a result.

In addition, a right-click action, **`Read Status Supported`**, is on the **`CcHrvStatus`** component. This allows you to force the reading of the HRV Status parameters supported by the device, where corresponding proxy points are added.

Note: *This is automatically done during the discovery process*

Proxy points

If the device supports the `HRV Status` command class, from 1 to 7 NumericPoints are added to its ZWaveDevice's **`Points`** container. Default names are according to available parameters (Table 5-4).

Table 5-4 *HRV Status proxy point default names to HRV status parameters*

Default proxy point name	HRV Status parameter
"outsideTemp"	Outdoor Air temperature
"supplyTemp"	Supply Air temperature
"exhaustTemp"	Exhaust Air temperature
"dischargeTemp"	Discharge Air temperature
"roomTemp"	Room temperature
"roomRelHum"	Relative Humidity in room
"remainingFilterLife"	Remaining filter life

zwave-CcIndicator

CcIndicator (**indicator**) is a child container of a ZWaveDevice object that supports the `Indicator` command class. Use it to read or change the "Indicator" value for the device, via the single read-only property "Indicator", as well as actions to read and/or set the Indicator on and off.

Note: *Depending on the device, "Indicator" behavior can vary. For example, this may control whether an LED stays lit on a wall-mounted device like a switch or dimmer, making it easier to find at night.*

Available actions on the CcIndicator (**indicator**) component are as follows:

- **Read** — Read from the device whether Indicator is On (true) or Off (false).
- **Indicator On** — Request the device to turn Indicator On.
- **Indicator Off** — Request the device to turn Indicator Off.

zwave-CcMultiInstance

CcMultiInstance (**multiInstance**) is a child container of a ZWaveDevice that supports either the `Multi Instance` (version 1) command class *or* `Multi Channel` (version 2) command class. It is used to control multiple instances of a given function in a device. In newer devices, the `Multi Instance` command class has largely been replaced by the similar `Multi Channel` command class, version 2.

Read-only child properties include the following:

- **Dynamic** — If true, indicates that the device has a dynamic number of instances, where the number of instances can change over time.
- **Identical** — If true, indicates that each instance supports the same command classes.
- **Number End Points**— Number of instances (end points) embedded in the device.

The driver also interrogates the device to determine the number of instances that exist for each of the supported command classes. See these instance counts on properties under the "Cmd Classes" component under the ZWaveDevice's "Node Info" component (NodeInformation), where they appear as "`count=1`" or "`count=2`", and so on.

If multiple instances are supported by the device, the driver may create additional components to model the additional instance. Any additional components are created using a name that includes a "`_n`" suffix, where `n` is the instance number.

Note: *The first instance does not have a suffix, that is, there is no "`_1`" suffix.*

For example, if the device supports two (2) instances of the Switch Binary command class, two Boolean-Writable proxy points are created: named "switch" and "switch_2."

Also, if the device supports the *Multi Channel* command class, additional "`assocInEp(n)`" slots are added to the ZWaveDevice component, to support associations to different endpoints.

zwave-CcProtection

CcProtection (**protection**) is a child container of a ZWaveDevice that supports the `Protection` command class, used to protect a device against unintentional control (e.g. a youngster). A single read-only child property ("Protection") indicates the current protection state.

Right-click actions on CcProtection (**protection**) are as follows:

- **Read** — Forces the driver to read the protection state from the device.
- **Set** — Provides a popup **Set** dialog, from which you can select a protection state to be used by the device, for example:
  - Unprotected

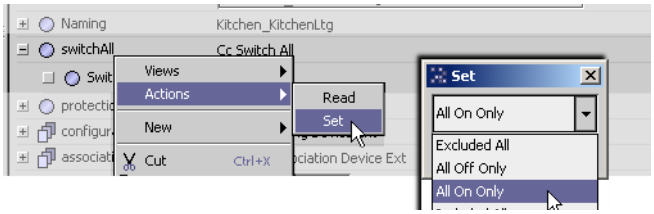
- Protect By Sequence
- Local Operation Blocked

zwave-CcSwitchAll

CcSwitchAll (**switchAll**) is a child container of a ZWaveDevice object that supports the `All Switch` command class. Its read-only properties specify how the Z-Wave "all devices" commands "All On" or "All Off" are handled, where this can vary among devices.

Actions "Read" and "Set" allow re-reading and changing the value in the device.

Figure 5-1 *Set actions on "switchAll" (CcSwitchAll) component*



The ZWaveNetwork has available "All On" and "All Off" actions that broadcast Z-Wave "all devices" commands. See "ZWaveNetwork actions" on page 3-11 for related details.

zwave-CcTime

CcTime (**time**) is a child container of a ZWaveDevice that supports the `Time` command class. It contains a single read-only "Date Time" property that reads the date and time from the device in the Z-Wave network.

zwave-CcWakeUp

CcWakeUp (**wakeUp**) is a child container of a ZWaveDevice that supports the `Wake Up` command class, typical for a *battery-powered* device. This allows the device to "sleep" for periods and then "wake up" occasionally to notify another device (always listening) that the device is ready to receive queued commands. Typically, (with the JACE operating as the SIS or SUC), this other device is the JACE.

Read-only properties of this component specify a number of parameters and statistics related to this wake up routine, as follows:

- **Is Awake** — Indicates by "true" that a wake-up notification has been received from the device, and the driver is actively reading or writing data from/to the device.
  Note: *There may be times that a device is actually awake, yet this property indicates "false". This can occur after a button on the device has been pushed, which typically keeps the device awake for a fixed period (for example, 10 minutes). The driver gets the notification when the button is pushed, and then immediately updates received data and executes any queued write messages. After completing all transactions, the driver then marks this property as false (typical after any "normal" awake routine). This is necessary because devices* do not *send any notification when a "button timeout" expires.*
- **Request** — Shows request status information. For example, if the component's "Set Wake Up Inteval" action is invoked to set it to 360 (seconds), the string "`setWakeUpInterval posted: 360`" would be shown. When the device wakes up and the interval is written, "`wakup interval written: 360 30-Jul-10 9:53 AM EDT`" would be shown.
- **Interval** — Reflects the time period, in seconds, that the device periodically wakes up and sends a Wake Up Notification command to the node specified in the "Node Id" property.
  Note: *A right-click "Set Wake Up Interval" action is available to modify this period, if needed.*
- **Min Interval** — Minimum wake up interval supported by this battery-powered device.
- **Max Interval** — Maximum wake up interval supported by this battery-powered device.
- **Default Interval** — Factory-default wake up interval for this battery-powered device.
- **Step Size** — Resolution of possible changes to the wake up interval for this battery-powered device.
- **Last Wake Up** — Timestamp the last Wake Up Notification message was received from this device.
- **Next Wake Up** — Timestamp of the next (anticipated) Wake Up Notification message to be received from this device. If two (2) consecutive wake up interval messages are missed from the device, the parent ZWaveDevice is flagged as "down".
- **Node Id** — The Z-Wave node ID to which this device sends its Wake Up Notification messages. If the JACE is acting as the network's SIS or SUC (recommended), it is automatically set to the JACE's node ID (2). Otherwise, it is set to 255, which is the Z-Wave broadcast address.

In addition to properties above, several *actions* are available on the CcWakeUp (**wakeUp**) component, described as follows:

- **Set Wake Up Interval** — Use to modify the wakeup interval used (and stored) in the device.
- **Post Read Node Info** — Forces the driver to request the device to send its Node Information message the next time the device wakes up. You might invoke this to be sure that all of its supported command classes are known to Niagara.
- **Read Device Data** — This action is automatically invoked when a Wake Up Notification message is received from the battery-powered device. It forces the driver to read the data from all supported command classes, including any device-resident components as well as any proxy (control) points.

Note: *For background details on battery-powered Z-Wave devices, see "Battery-powered (non-listening) device considerations" on page 1-5.*

zwave-NodeInformation

NodeInformation (**Node Info**) is a child container of any ZWaveDevice component (ZWaveDevice or ZWaveThermostat). Its read-only properties specify all of the device classes and command classes supported, as returned by the Z-Wave device. An action is also available.

For more details, see "Node Info" on page 3-16.

zwave-UserInfo

UserInfo (**usern**) is a child container under a ZWaveDevice's "**users**" component (ZWaveUserDeviceExt), available only if the device supports the `User Code` command class. It contains information about a local Z-Wave user in the device.

Read-only properties of a user are as follows.

- **User Id** — Integer value ID, unique among all users of device.
- **User Id Status** — Numerical status of the user.
- **User Code** — String to associate with the user.

zwave-ZWaveAssociationDeviceExt

ZWaveAssociationDeviceExt (**association**) is a child container of a ZWaveDevice, providing the device represented supports the `Association` command class. Associations are application-level connections between devices, where each is represented by a child ZWaveAssociationGroup component.

Typically, you graphically view and create such associations between ZWaveDevices by working from the wire sheet of the ZWaveNetwork. Read-only properties of this component include the following:

- **Version** — The version of this command class.
- **Is Multi Instance** — Indicates if multiple instance associations are supported (true) or not (false).
- **Supported Groupings** — Indicates the number of groupings this node supports. A grouping can be considered the "source" of an association.

For related details, see "About Z-Wave Associations" on page 3-30.

zwave-ZWaveAssociationGroup

ZWaveAssociationGroup (**groupn**) represents an association to one or more Z-Wave devices, and and is a child of a ZWaveDevice's **associations** extension (ZWaveAssociationDeviceExt). Each group contains a number of read-only properties, including the following:

- **Request** — Typically blank, but may have text if a pending request to battery-powered node.
- **Group Id** — Integer number of association group.
- **Max Nodes Supported** — Maximum nodes that can be associated from this group.
- **Nodes** — Node IDs of association targets.
- **Nodes String**— Same as above.
- **Multi Instance Nodes** — Node IDs of multi instance association targets.
- **Multi Instance Nodes String** — Same as above

By default, an association to the JACE's Z-Wave interface (Z-Wave option card or serial Z-Wave gateway) is automatically assumed. Typically, other associations are added by working from the wire sheet view of the ZWaveNetwork. Adding wire sheet links results in child group1 - group*n* components.

For more details, see "About Z-Wave Associations" on page 3-30.

zwave-ZWaveConfigDeviceExt

ZWaveConfigDeviceExt (**configuration**) is a child container of a ZWaveDevice that supports the the Configuration command class. Use it to retrieve (and if necessary, edit) various "factory default" settings in the selected Z-Wave device. Included is a right-click *action* to "Read Config Data", which dynamically adds writable config1 - config*n* properties, each with the current integer value.

The following properties are always included:

- **Max Number Config Properties** — Maximum number of configuration parameters the driver attempts to retrieve when invoking the "Read Config Data" action. The default is 10, and is adjustable. If greater than the available number of config parameters, upon a "Read Config Data" action, after all available parameters are retrieved, a "no response to config*n* request" message appears in the "Response" property, and further requests cease.
- **Request** — (read-only) Reflects in real time the current request, e.g. "configData 2 requested".
- **Response** — (read-only) Reflects the current or last results from the last "Read Config Data" action. Included are "no response" messages if too many parameters were attempted (see above), as well as messages related to pending operations if a battery-powered device.

Following a successful "Read Config Data" action, one or more writable properties are included:

Note: *All have integer values from 0 - 255.* Before modifying *any of these properties, refer to the documentation for that Z-Wave device to be sure the results will be as intended.*

- **config1** — First of the device's available configuration parameters.
- **config*n*** — Next in the series of available configuration parameters.

Modifying and saving a config property value causes the new value to be written to the device.

Note: *If the device is battery-powered and is "sleeping", any read request ("Read Config Data" action) or write request (modified config property) is posted for execution the next time that the device wakes up. This is reflected in the "Request" and "Response" status properties, described above.*

zwave-ZWaveDevice

ZWaveDevice is one of two "device-level" components in a ZWaveNetwork, and represents a specific node (the other device-level component type is the ZWaveThermostat). Each has a Points device extension (ZWavePointDeviceExt) that contains all proxy points for polling.

A ZWaveDevice has the standard device component properties such as status and enabled (see "Common device components" in the *Drivers Guide* for general information). Common Z-Wave device properties include its unique Node Id, Node Info, and Device Profile.

Depending on the specific Z-Wave command class(es) supported by the device, one or more command class containers are dynamically added to each ZWaveDevice. Each command class container contains specific properties, and in some cases may contain child components. Some command classes also result in proxy (control) points created under the Points extension of the ZWaveDevice.

The default view for a ZWaveDevice is its property sheet. A number of actions are also typically available. For more details, see "About the ZWaveDevice" on page 3-14.

zwave-ZWaveManufacturerSpecific

ZWaveManufacturerSpecific (**manufacturerSpecific**) is a child container of a ZWaveDevice that supports the Manufacturer Specific command class. Its read-only properties reflect assignments for specific ID values stored in the device, as follows:

- **Manufacturer Id** — Unique manufacturer ID, as assigned by Zensys.
- **Product Type Id** — Product type ID, assigned by the manufacturer.
- **Product Id** — Product ID, assigned by the manufacturer.

Values use hexadecimal notation. For example, a device might have:

- Manufacturer Id = 1a
- Product Type Id = 5244
- Product Id = 2

In Workbench, these ID values are used in the file name for a saved "device template" (Device Profile), to associate with the correct device. The example device (above) would have a device profile file name of:

```
x1a_x5244_x2.info
```

For related details, see "About Device Profiles" on page 3-32.

zwave-ZWaveNames

ZWaveNames (**Naming**) is a child container of a ZWaveDevice, providing it supports the `Node Naming and Location` command class. It is used to assign and store name and location text strings in the device's non-volatile memory.

This component has read-only properties that reflect the assigned "Node Name" and "Node Location" of the Z-Wave device. Included are the following right-click *actions*:

- **Assign Node Name**— Provides a popup field (blank) in which you can enter a Node Name to write to the device.
- **Assign Node Location** — Provides a popup field (blank) in which you can enter a Node Location to write to the device.
- **Read Node Name**— Forces the driver to read the Node Name from the device.
- **Read Node Location** — Forces the driver to read the Node Location from the device.

Note: *The two "Assign" actions (to write to the device) also appear as actions on the parent ZWaveDevice component. See "ZWaveDevice actions" on page 3-19.*

zwave-ZWaveNetwork

ZWaveNetwork is the top-level component for the Z-Wave driver in a station. It provides configuration parameters necessary for the driver to communicate with a network of Z-Wave devices through a Z-Wave serial gateway—either a JACE controller's Z-Wave option card, or an external, third-party, serial Z-Wave gateway. The ZWaveNetwork component has the typical collection of slots and properties as most other network components—refer to "Common network components" in the *Drivers Guide*.

The ZWaveNetwork also has a number of properties and actions unique to operation in a Z-Wave system. For more details, see "About the ZWaveNetwork" on page 3-7.

zwave-ZWaveNetworkScene

ZWaveNetworkScene represents a Z-Wave *scene*, and is a child of the Scenes network extension (ZWaveSceneNetworkExt) of the ZWaveNetwork. Each scene has a unique numerical "Scene Id" (from 1 - 255), assigned by the system. Each scene can contain multiple scene "config" containers, which contain properties that define how different nodes are controlled when the scene is activated.

The default view for any scene is the **Network Scene Manager**, which you use to add or edit scenes configs, discover nodes that support scenes, or activate scenes. For more details, see "About Z-Wave Scenes" on page 3-27.

zwave-ZWaveNetworkSceneConfig

ZWaveNetworkSceneConfig is a container that specifies how one node is affected by a specific Z-Wave *scene*, and is a child of a scene (ZWaveNetworkScene). It holds configuration properties that include the level and duration for dimming (if applicable for control of a dimmer node), as follows:

- **Scene Id** — Unique integer 1 - 255, to specify the scene (automatically assigned by the system).
- **Dimming Duration** — Specifies the time, in Duration Units, that a dimming device takes to ramp to the specified scene Level.
- **Duration Units** — Specifies time units for Dimming Duration.
- **Override** — If set false, then the current level or state of the device is uses as the dimming level, and the following "Level" property is ignored.
- **Level** — Specifies the final dim level, from 0 - 100. If this is a Boolean (binary) device, then 0 = Off, 100 (or any non-zero value)= On.

You typically use the default view of the parent scene, in the Network Scene Manager, to add or edit these scene configs. For more details, see "Creating and setting up scenes" on page 3-27.

zwave-ZWavePointDeviceExt

ZWavePointDeviceExt (default name `Points`) is the container for ZWave proxy points under a Z-Wave device object (ZWaveDevice or ZWaveThermostat). Proxy points represent data items in the device that need to be polled for data. It operates as in most other drivers; see "About the Points extension" in the *Drivers Guide* for general information. The default and primary view for the Points extension is the **ZWave Point Manager**.

For specific Z-Wave details, see "ZWave Point Manager" on page 3-20 and "About ZWave proxy points" on page 3-21.

zwave-ZWaveProxyExt

ZWaveProxyExt is the most common proxy extension for any dynamically-created ZWave proxy point. It represents a single piece of data defined by a specific Z-Wave command class. It has standard proxy extension properties such as Status and Enabled, among others. For additional details, see "About ZWave proxy points" on page 3-21 and "ZWave proxy points by default name" on page 3-23.

zwave-ZWaveSceneActuatorConf

ZWaveSceneActuatorConf (**scenen**) represents the action configuration for this device during a specific *scene*, and and is a child of a ZWaveDevice's **scenes** extension (ZWaveSceneDeviceExt). Read-only properties reflect the current configuration, which if necessary can be edited from the **Network Scene Manager** view of a **scene** under the **Scenes** network device extension. For related details, see "About Z-Wave Scenes" on page 3-27.

zwave-ZWaveSceneDeviceExt

ZWaveSceneDeviceExt (**scenes**) is a container slot on a ZWaveDevice, providing the Z-Wave device represented supports *scenes*. It has no configuration or status properties. If one or more scenes have been defined that include this device, it contains corresponding child "scene actuator configuration" (ZWaveSceneActuatorConf) components.

Although scenes have this "device-level" visibilty, they are typically engineered at the "network level", using the **Scenes** network extension on the Z-WaveNetwork. For more details, see "About Z-Wave Associations" on page 3-30.

zwave-ZWaveSceneNetworkExt

ZWaveSceneNetworkExt (**Scenes**) is a frozen container slot on a ZWaveNetwork, used to manage (create, edit and hold) Z-Wave *scenes*. It has no configuration or status properties.

The default view for Scenes is the **Network Scene Ext Manager**, which you use to add New scenes, edit existing scenes, or activate scenes. For more details, see "About Z-Wave Scenes" on page 3-27.

zwave-ZWaveSerialPlugin

ZWaveSerialPlugin (**Comm Plug In**) is a container on a ZWaveNetwork, used to configure serial parameters and other messaging parameters for the host's Z-Wave interface (JACE Z-Wave option card or external Z-Wave serial gateway). For related details, see "Comm Plug In" on page 3-8.

zwave-ZWaveThermostat

ZWaveThermostat is one of two "device-level" components in a ZWaveNetwork, and represents a specific thermostat node (the other device-level component type is the ZWaveDevice). Each has a Points device extension (ZWavePointDeviceExt) that contains all proxy points for polling.

A ZWaveThermostat has the standard device component properties such as status and enabled (see "Common device components" in the *Drivers Guide* for general information). Common Z-Wave device properties include its unique Node Id, Node Info, and Device Profile. Depending on the specific Z-Wave command class(es) supported by the device, one or more command class containers are dynamically added to each ZWaveThermostat, and proxy (control) points are also created.

The default view for a ZWaveThermostat is its property sheet. A number of actions are also typically available. For more details, see "About the ZWaveDevice" on page 3-14 and "Types of ZWave device components" on page 3-14.

zwave-ZWaveThermostatSetPointProxyExt

ZWaveThermostatSetPointProxyExt is the proxy extension for a dynamically-created ZWave proxy point for a setpoint in a ZWaveThermostat device component. It represents a single setpoint defined by the Z-Wave Thermostat Setpoint command class. It has standard proxy extension properties such as Status and Enabled, among others. For additional details, see "About ZWave proxy points" on page 3-21 and "Thermostat Setpoint" on page 3-26.

zwave-ZWaveUserDeviceExt

ZWaveUserDeviceExt (**users**) is a child container of a ZWaveDevice, providing the Z-Wave device represented supports the `User Code` command class. Users are application-level users of a device, where each is represented by a child UserInfo component.

A single read-only property is as follows:

- **Number Users** — Total number of defined users.

zwave-ZWaveVersion

ZWaveVersion (**`Version`**) is a child container of a ZWaveDevice, providing it supports the `Version` command class. Its read-only properties specify the following:

- **Library Type** — Numerical Z-Wave library type.
- **Protocol Version** — Numerical Z-Wave protocol version.
- **Application Version** — Numerical vendor-specific application version.

A right-click *action* "Read Cmd Class Versions" is also available, which is automatically invoked upon a device discover. This retrieves the information above, as well as the version of each of the device's supported command classes. These versions are reflected in the ZWaveDevice's "Node Info", "Cmd Classes" properties (see "Node Info" on page 3-16).