

Information and specifications published here are current as of the date of publication of this document. Tridium, Inc. reserves the right to change or modify specifications without prior notice. The latest product specifications can be found by contacting our corporate headquarters, Richmond, Virginia. Products or features contained herein may be covered by one or more U.S. or foreign patents. © 2009 Tridium, Inc.

Niagara R2 to NiagaraAX via oBIX

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

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

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

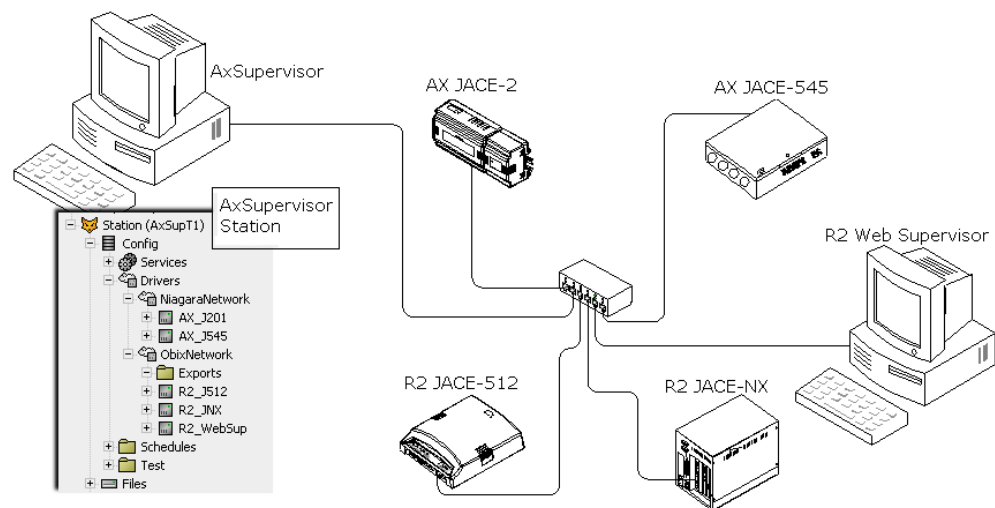
The following main sections are in this document:

- “R2 to AX oBIX Overview” on page 2-1
- “R2 station engineering” on page 2-3
- “AxSupervisor engineering” on page 2-7
- “Document change log” on page 2-23

R2 to AX oBIX Overview

In NiagaraAX-3.1 and later, an AX station can use the obixDriver for client-side oBIX access of remote R2 stations to mix R2 and AX data together. In the most-anticipated scenario, an AxSupervisor will be added to a job that already has R2 JACEs and an R2 Web Supervisor, along with additional AX JACEs. A possible eventual goal of the AxSupervisor is to replace the R2 Web Supervisor. This document focuses on the AxSupervisor, and notes areas where R2 to AX “switchover” has limitations, or may require additional engineering.

Figure 2-1 AxSupervisor access to R2 stations and AX JACEs is using different network types



Requirements

- Each R2 host (JACE, Web Supervisor) should be running Niagara 2.301.522 or later, have the obix module 2.301.527c.beta or later installed, and has its license enabled for the “obix” feature. Your JDE (WorkPlace Pro) should also be at 2.301.522 or later.
Optionally, each R2 host can also have an “obixInternal.properties” file in its nre\lib folder. This helps when doing Obix proxy point discovers from the AX client side, in coordination with an AX feature to hide rarely accessed (internal) properties. See [“About Discover ‘Include’ options”](#) on page 2-10 for related details.
- The AxSupervisor host must be running NiagaraAX 3.1.31 or later, have the installed modules:
 - obix 3.3.22 or later
 - obixDriver 3.3.28.2 or laterThe AxSupervisor also requires the “obixDriver” feature in its license. Any AX JACE that needs direct Obix client access to any R2 JACE also has the same requirements.

Loading effects of oBIX integration

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

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

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

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

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

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

Engineering summary

Most engineering is expected to occur in the AxSupervisor station (see [Figure 2-1](#) on page 1), where:

- All data from remote R2 stations is modeled under the Drivers/ObixNetwork, using the basic familiar AX driver architecture (e.g. network/devices/Points/proxyPoints). See [“AX station engineering summary”](#).
- All data from remote AX JACEs is modeled under the Drivers/NiagaraNetwork, as for any AxSupervisor. Refer to [“About the Niagara Network”](#) in the *NiagaraAX Drivers Guide* for more details.

R2 station engineering summary

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

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

For more details, see [“R2 station engineering”](#).

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

AX station engineering summary

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

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

For more details, see [“AxSupervisor engineering”](#) on page 2-7.

R2 station engineering

The following sections apply to R2 station and host engineering:

- [Pre-integration R2 station changes](#)
- [Add ObixService](#)
- [R2 ObixExport objects](#)
- [Other possible R2 host changes](#)

Pre-integration R2 station changes

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

- [Reducing R2 station resource count](#)
- [Reducing R2 swid lengths](#)

Reducing R2 station resource count

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

Reducing R2 swid lengths

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

Example

Original swid:

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

Shortened swid:

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

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

Add ObixService

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

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

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

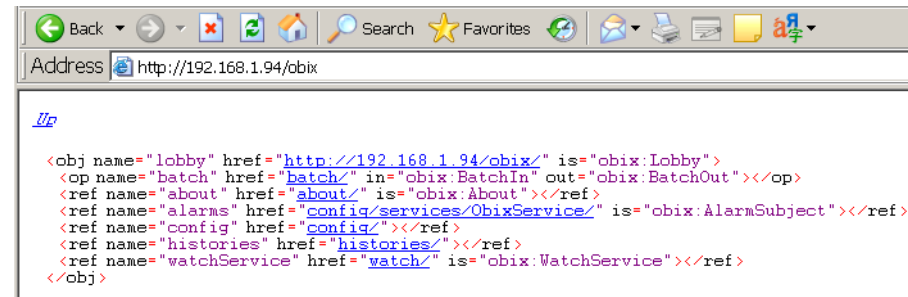
where `<host>` is IP address or hostname, and `[:port]` is optional (if omitted, assumed as 80).

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

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

As shown in [Figure 2-2](#), after you login with station credentials you see an HTML representation of the station's oBIX lobby, including hyperlinks to traverse into the object tree structure.

Figure 2-2 Example browser connection to confirm R2 station oBIX server operation



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

Note: For this reason, the AxSupervisor station’s “Exports” folder under its ObixNetwork is not used when integrating Niagara R2 stations as ObixClients. However, AX to R2 schedule exports are possible using a different method—see “R2ObixClient R2 Schedule exports” on page 2-22.

Additional R2 station engineering topics are explained in the following sections:

- R2 ObixExport objects
- Other possible R2 station configuration changes

R2 ObixExport objects

Note: Use of these objects is entirely optional. Many R2 to AX oBIX integrations have not used them, as standard AX Obix proxy points for these specific R2 object types provide “right click command access”, along with value and status. Use is necessary only to permit an “interstation control link”, from AX to the R2 station. The three “export” objects in the tridiumx/obix jar are available if you want to allow “link control” writes from AX to R2 objects, for example to the “priorityArray” input of an AnalogOutput, BinaryOutput, or MultistateOutput object. In this case, you copy one of objects from the R2 tridiumx/obix jar and paste it into the station, linking its output into the priorityArray input of the R2 object being controlled.

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

Figure 2-3 Example ObixAnalogExport object copied into station for linking into AnalogOutput object

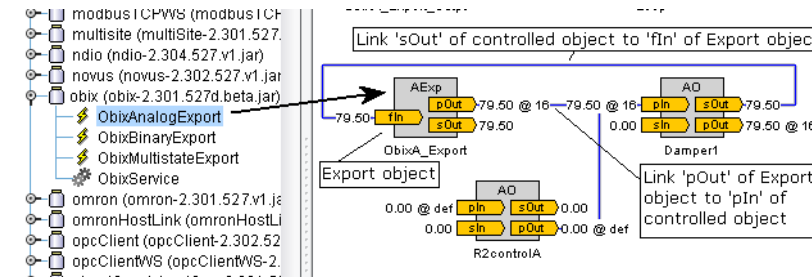


Figure 2-3 shows an example of both links, where an ObixAnalogExport object controls an AnalogOutput (“Damper1”).

Figure 2-4 ObixBinaryExport object copied into station for linking into BinaryOutput (BO) object

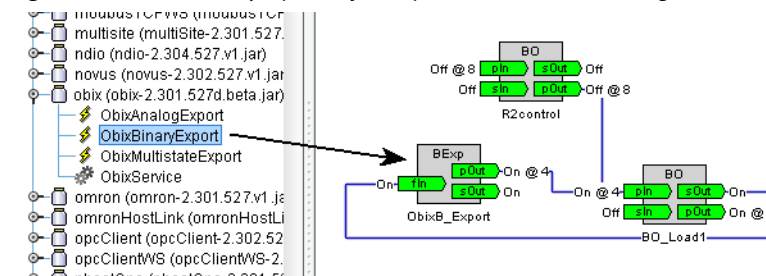


Figure 2-4 shows a similarly-linked ObixBinaryExport object used for linking into a BinaryOutput object, “BO_Load1”.

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

Figure 2-5 ObixAnalogExport object copied into station for use as “statusInput” type value

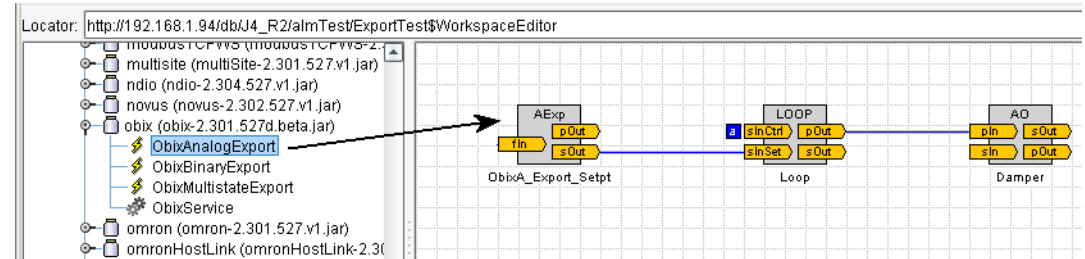


Figure 2-5 shows an example where an ObixAnalogExport object is used for setpoint control of an R2 Loop object, linked into the Loop’s “sInSet” input.

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

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

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

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

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

- priority — The priority to be used for writing values at the prioritizedOutput (defaults to 16).
- units (if ObixAnalogExport), or
activeInactiveText (if ObixBinaryExport), or
stateText (if ObixMultistateExport) — In any type of export object, set units, etc. to mirror the units for the linked input on the controlled R2 object.

Note: An ObixAnalogExport object has these additional Config properties, described below (see [Figure 2-6](#)). Use of these “limit” type properties is optional.

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

Figure 2-6 Config properties for ObixAnalogExport, including “limit” type properties

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

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

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

Other possible R2 station configuration changes

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

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

- If mastering R2 Schedule objects from an AxSupervisor (or another AX JACE), and they are currently “slaved” to another R2 Schedule, you need to first *unlink* those slaved R2 Schedules (remove externalSubscription at “slaveIn” input). For more information on engineering for the “AX side,” see [“AxSupervisor engineering”](#) on page 2-7.
- Depending on how log archiving is to continue for any R2 JACE station, you may wish to change the setup of the station’s LogService, with the following Config tab properties:
 - archiveMode — from “archive_remote” to “archive_local”.
 - archiveAddress — uncheck Supervisor entryFor related details on archiving R2 logs as histories to an AxSupervisor, see [“R2ObixClient Histories \(logs and archives\)”](#) on page 2-16.
- Depending on alarm/alert management is to continue for any R2 JACE station, you may wish to change the setup of the station’s NotificationService, with the following Config tab properties:
 - alarmArchiveAddress — uncheck Supervisor entry
 - archiveMode — from “archiveRemote” to “archive_local” (if a JACE-4/5, “archive_local_no_SQL”).

For related details on R2 alarms/alerts at an AxSupervisor, see [“R2ObixClient Alarms”](#) on page 2-18.

Other possible R2 host changes

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

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

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

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

```
webServer.threadPoolSize=n
```

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

AxSupervisor engineering

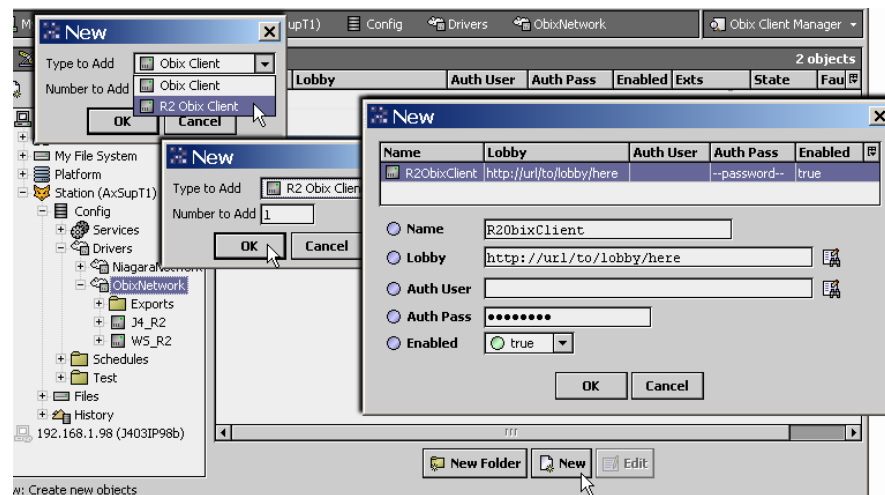
The following sections describe AxSupervisor engineering topics to integrate R2 stations:

- [ObixNetwork and R2ObixClient devices](#)
- [R2ObixClient Points](#)
- [R2ObixClient Histories \(logs and archives\)](#)
- [R2ObixClient Alarms](#)
- [R2ObixClient R2 Schedule exports](#)

ObixNetwork and R2ObixClient devices

In the AxSupervisor you add a single ObixNetwork under Drivers, then manually add R2ObixClient devices, where each represents an R2 station. For each new R2ObixClient, you enter a few properties in the **New** dialog, shown with non-working default values in [Figure 2-7](#).

Figure 2-7 Add R2ObixClients using New button in Obix Client Manager view of ObixNetwork



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

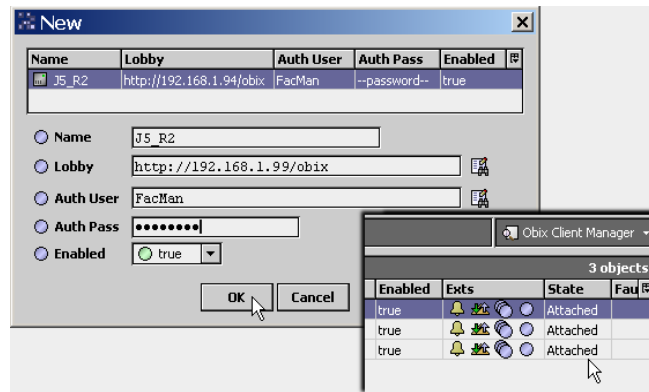
Also, note the general “client/server” naming in NiagaraAX follows a “convention” used in some other drivers, for example the OPC driver and Modbus drivers, where a “client” device actually represents a server device (here, an oBIX server), and associated NiagaraAX components are named “client” because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

Set properties in the New dialog as follows:

- **Name**
The AX name for the device component—by convention, enter the R2 station name. It must be unique among other child devices, and if using history ID defaults, also recommended to be unique from any NiagaraStation names (under the station’s NiagaraNetwork, for remote AX stations).
- **Lobby**
The URI to the root of the R2 station’s oBIX server object tree (lobby), using syntax:
`http://<host>[:port]/obix`
where <host> is IP address or hostname, and [:port] is optional (if omitted, assumed as 80).
For example: `http://192.168.1.94/obix` for a typically-configured station at that IP address, or `http://192.168.1.75:85/obix` for a station running on httpPort 85 on a host at that IP address.
- **Auth User**
Enter user name in that R2 station, typically with all admin-level privileges for most security groups.
Note: On the NiagaraAX client side, if you attempt object writes without this user having the necessary security rights for those objects (for example a command, or modify a property), it results in an “HTTP Error 401:Access Denied” error.
- **Auth Pass**
Enter password for that user account in the R2 station.
- **Enabled**
Defaults to true (must be true to attempt communications and operate).

After entering data and clicking OK, the device is added to the database, and the “State” column value for the new row quickly changes from “Detached” to “Attaching”, and finally to “Attached”, as in [Figure 2-8](#).

Figure 2-8 Entering new ObixClient and subsequent State change to Attached



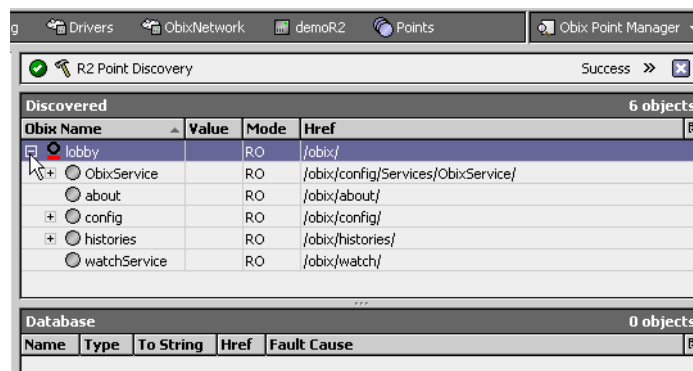
Note: (Troubleshooting) If after entering the R2ObixClient it remains Detached, review its Lobby syntax, including IP address of the R2 JACE (open a command prompt window and issue a Ping command to that IP address). Also, verify the R2 station user credentials for Auth User and Auth Pass are correct. Note also that you should be able to open a browser connection to the R2 station using the URL entered for Lobby, and after entering user credentials, see its oBIX lobby (see [Figure 2-2](#) on page 4).

R2ObixClient Points

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

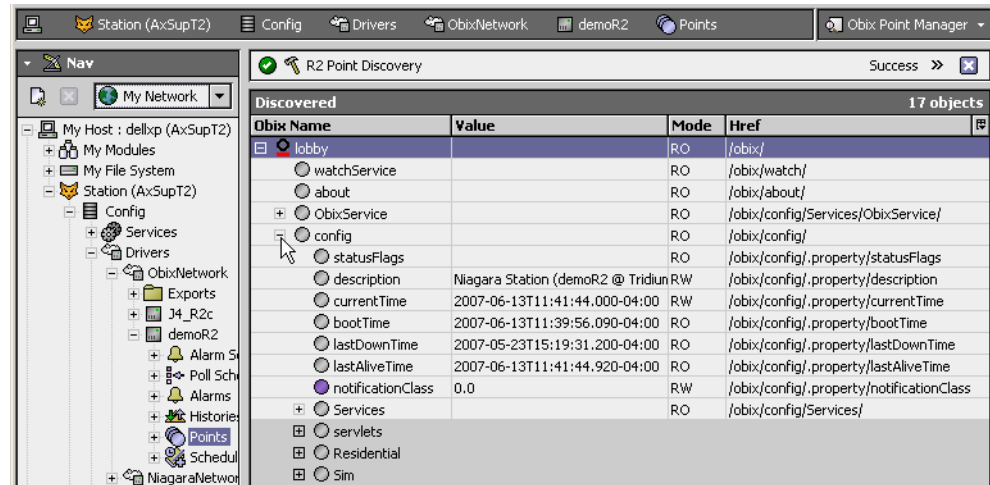
In the discovered pane of the Obix Point Manager, expand the root “lobby” to see the tree organization, as shown in [Figure 2-9](#). Items of practical interest for proxy points are under the “**config**” branch.

Figure 2-9 Top-level lobby structure in R2 station



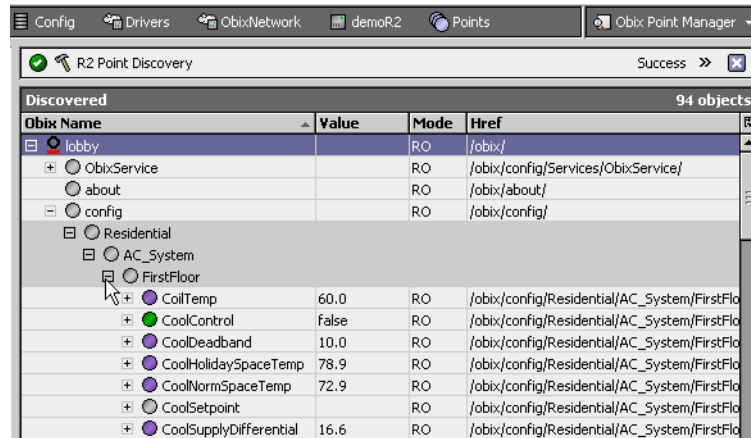
Expand the “config” branch of the lobby to find proxy point candidates. The config hierarchy reflects the R2 station’s object hierarchy, including a row for each object *property*, and expandable rows for child objects. The highest-level config node is the Station object. So when you first expand config, at the top are the properties found on the various tabs of the property sheet for the *Station* (when using the R2 JDE). See [Figure 2-10](#).

Figure 2-10 ObixPointManager with lobby expanded to show discovered proxy point candidates



Note: As needed, click on column headers in the Discovered pane to sort as ascending ▲ or descending ▼, in ASCII character order. For example, if you have the Obix Name column sorted ascending, when you expand items containers will be at top, and properties at bottom, as shown in [Figure 2-11](#).

Figure 2-11 Use column header sorts and expand containers as needed to find R2 objects and properties



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

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

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

See the following sections for more details:

- [About Discover “Include” options](#)
- [Specify the property](#) (important)
- [Add notes for R2 Obix proxy points](#)
- [About Obix proxy points](#)
- [About forceUpdate](#)
- [Command Notes](#)
- [Link control into R2 objects](#)
- [AnalogExport Limit Notes](#)
- [ObixClient proxy point tips](#)

About Discover “Include” options

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

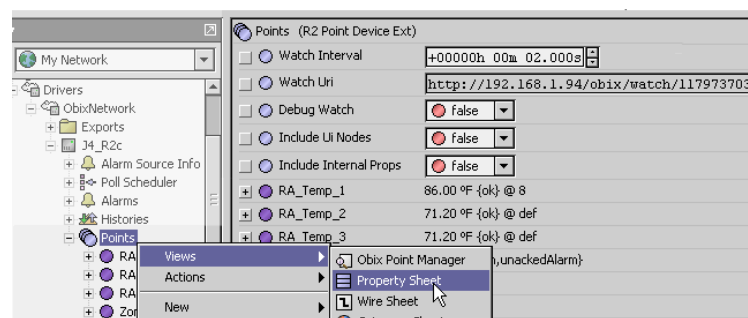
- All R2 container-type objects, including types Container, Bundle, PollOnDemand, and PollAlways appear in the discovered lobby tree as “groups” (gray rows)—you still expand them to see child objects. If you double-click (to add), an Obix Point Folder is immediately created, without an intervening popup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

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

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

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

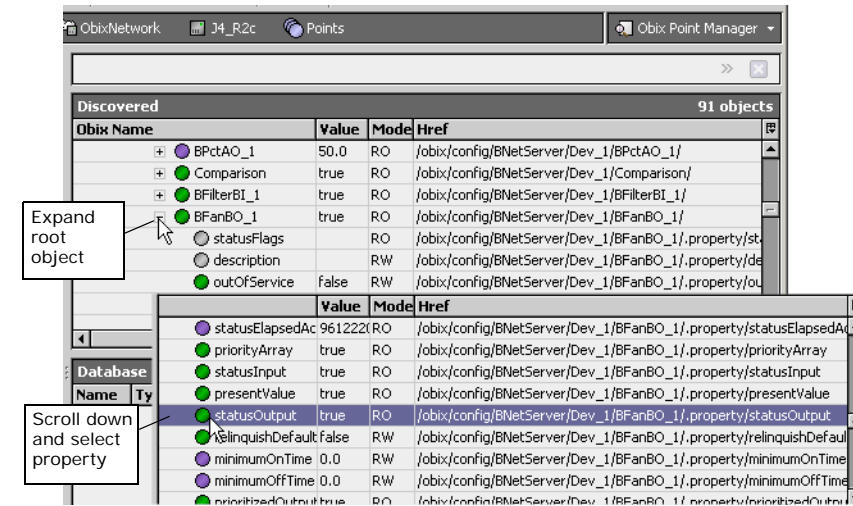
Figure 2-12 R2PointsDeviceExt property sheet with Include defaults



Specify the property

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

Figure 2-13 Select property of a discovered R2 object, vs. root (Node) object



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

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

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

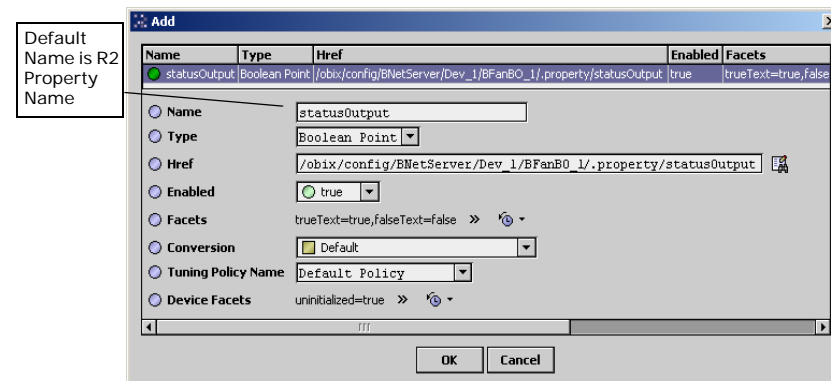
In summary, selecting object properties (rather than their root objects) will have a big impact on performance of the oBIX integration. Typically, an order of magnitude or two can be gained by specifying a property for each proxy point, versus specifying the root object.

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

Add notes for R2 Obix proxy points

When adding R2 Obix proxy points from the discovered lobby, and selecting a property (recommended), note that currently the default **Name** is the same as the selected R2 property name, such as “statusOutput”, “prioritizedOutput”, and so on. In addition, Facets show default values. See Figure 2-14.

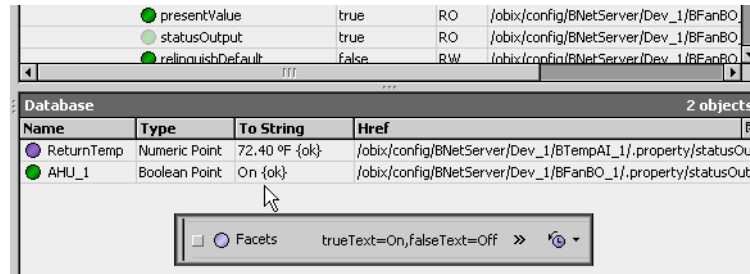
Figure 2-14 Example Add dialog for R2 Obix proxy point



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

Facets in the **Add** dialog do *not* need editing, even though uninitialized values (sometimes “null”) are shown. Upon adding the proxy point to the station, the units (or activeInactiveText) in the source R2 object are automatically uploaded and used as the Facets in the proxy point, as shown in [Figure 2-15](#).

Figure 2-15 Facets are automatically learned from the R2 object’s units or activeInactiveText



See the next section [“About Obix proxy points”](#) for additional details.

About Obix proxy points

Added Obix proxy points often resemble Niagara proxy points (under Points container of a NiagaraStation in a NiagaraNetwork) in the following ways:

- Most proxy points for R2 objects, including ones for “input writable” ones such as BinaryOutput, AnalogOutput, MultistateOutput, Loop, and so forth are proxied as *read-only* points only—writable AX point types like BooleanWritable, NumericWritable, etc. are not selectable. This applies to all nodes and properties that appear in the Discovered table with a “Mode” of “RO” (read only). However, object *commands* are available, as actions of the read-only proxy point. See [Figure 2-16](#) and also [“Command Notes”](#) on page 2-13.
- Some R2 object properties show a discovered “Mode” of “RW”. Most are configuration types such as alarm limits (e.g. “lowLimit”, “highLimit”), “deadband”, and “notificationClass” as a few examples. If desired, you can proxy such a property as a *writable* point type, e.g. NumericWritable, BooleanWritable, etc. This allows you to write the R2 property value from the AX station, either by an invoked action on the proxy point, or by linking to the proxy point’s input(s).

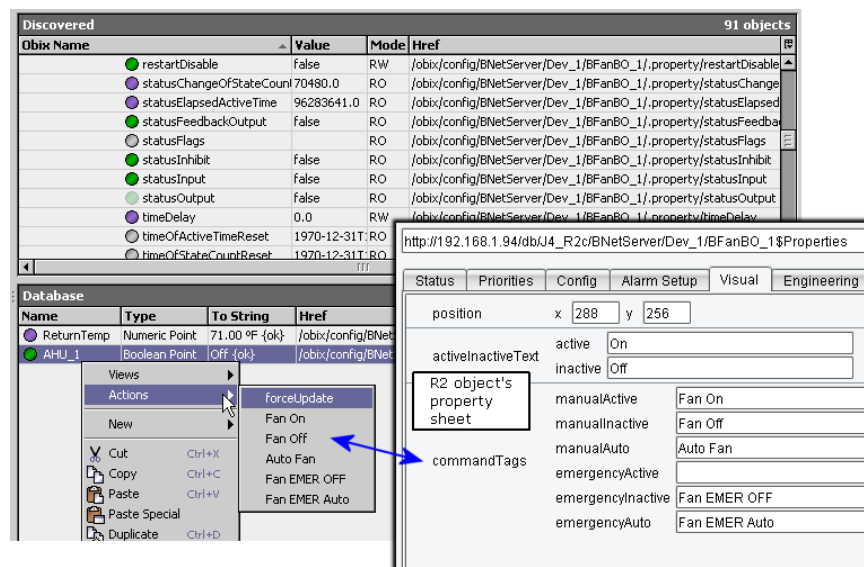
Note: By default, right-click actions on a writable Obix proxy point include both

- native AX actions for the writable point - to change the value of the specific R2 *property*
- actions for the commands on the parent R2 object - to directly command that object

In some cases, you may wish to *hide* some action slots, e.g. ones for the parent object’s commands. Or, if link control (via proxy input) is the only intended method, you may wish to *hide all* action slots. If hiding any actions, be sure to hide the “forceUpdate” one. See [“About forceUpdate”](#) on page 2-13.

Typically, “standard” control logic linking from AX to an R2 object requires additional engineering on both sides; see [“Link control into R2 objects”](#) on page 2-14.

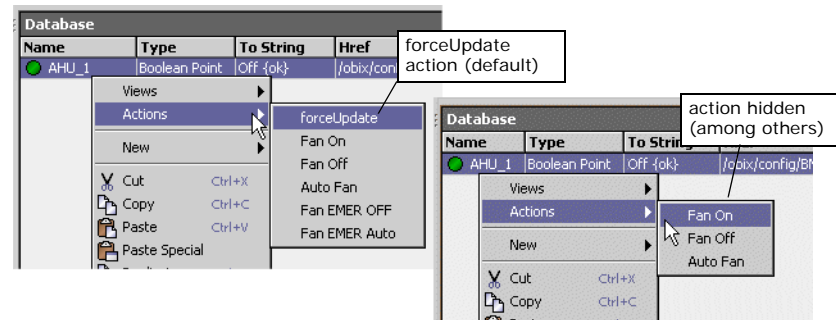
Figure 2-16 Proxy point for R2 object or property offers actions for commands, even as read-only point



About forceUpdate

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

Figure 2-17 Each Obix proxy point has forceUpdate action available by default



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

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

Command Notes

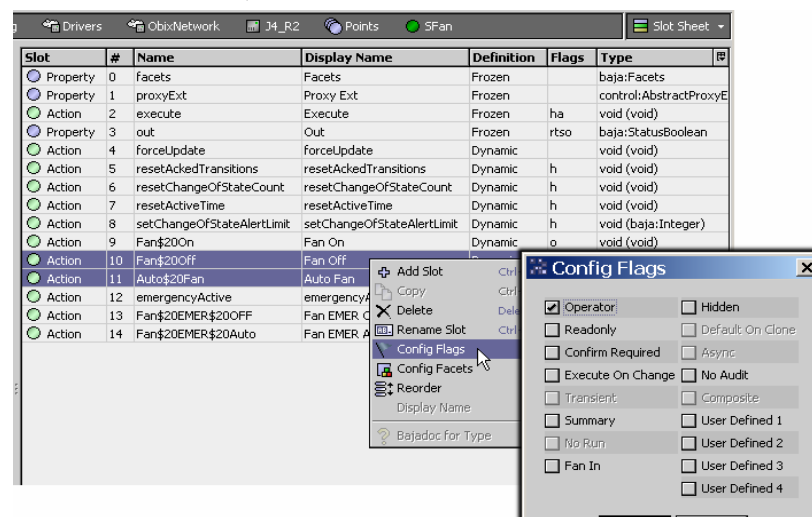
In addition to a proxy point “forceUpdate” action, note by default “normal right-click” commands appear on the action menu for Obix proxy points—providing the source R2 object has commands. Actions display mirroring the configured R2 object’s “commandTags” property strings, where applicable.

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

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

Note: Actions for R2 object commands that are operator level (“Cmd, Std”) automatically default with the *Operator* config flag set. See [Figure 2-18](#) for how these defaults look from the slot sheet of a proxy point.

Figure 2-18 Obix proxy point from its slot sheet, showing Operator flag set on manual-level actions



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

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

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

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

Examples of such commands for various control objects include:

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

If needed, you can expose any of these type actions by going to the slot sheet of the Obix proxy point, and clearing the “Hidden” config flag. As shown in the [Figure 2-18](#) slot sheet, these hidden actions appear listed with an “n” in the Flags column. However, it is anticipated that typically such actions will be left hidden. Note if hiding or unhiding command actions, you should hide the [forceUpdate](#) action too.

Link control into R2 objects

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

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

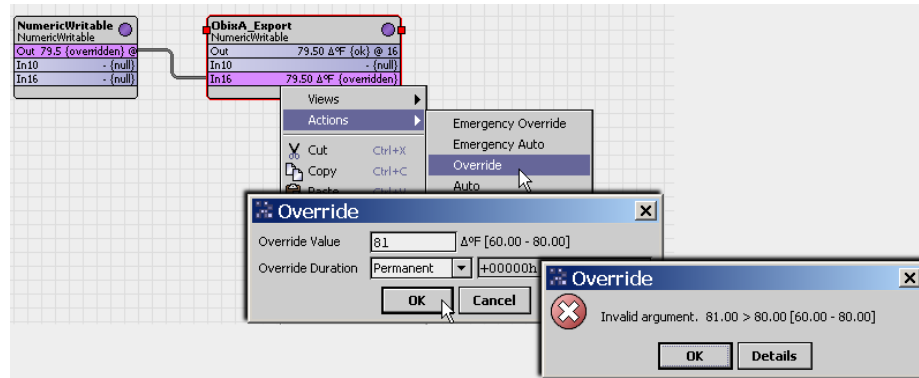
Then, back in the AX station with the ObixClient, you rediscover the Points (“lobby” object tree), and add a new proxy point for the “root node” of each added export object, selecting the default writable point (NumericWritable, BooleanWritable, etc.) for each one.

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

AnalogExport Limit Notes

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

Figure 2-19 Invalid Argument popup error in AX Workbench if invoking action outside of limit range.



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

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

ObixClient proxy point tips

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

- As needed, use the column sort features when traversing the lobby tree in the Discovered pane. This can save scrolling time.
- When creating ObixPointFolders for organizing proxy points (double-clicking R2 container type objects, or using the **New Folder** button), be aware of the current database level. Each ObixPointFolder has its own Obix Point Manager view, seen when you double-click it—note this collapses the lobby tree in the discovered pane. However, you can simply re-expand the lobby to find and add child Obix proxy points.
- For any R2 object for which you need standard AX alarming, create a root-level proxy point for it, then add an AX alarm extension to it (for example, an OutOfRangeAlarmExt, duplicating the same alarm and fault (min/maxPresentValue) limits as in the R2 source object). Or, to avoid limits duplication in AX, you can add a StatusAlarmExt and simply select the alarm states in the algorithms for OffNormal and Fault.
Note: Alternatively, you can configure the Alarms device extension under the R2ObixClient to process R2 native alarms within the AX station's alarm subsystem. This works best for R2 objects that are already being proxied in the AX station. For more details, see [“R2ObixClient Alarms”](#) on page 2-18.
- Understand the “forceUpdate” action of an Obix proxy point can cause issues in certain cases. For details, see [“About forceUpdate”](#) on page 2-13.

Client polling timing

As with most drivers, the polling cycle timing is dependent on the number of proxy points currently in the “Dibs”, “Fast”, “Normal” and “Slow” polls. Assuming the server device is an R2 station or a device that supports the watch subscription, the “Watch Interval” property controls the polling of the watch.

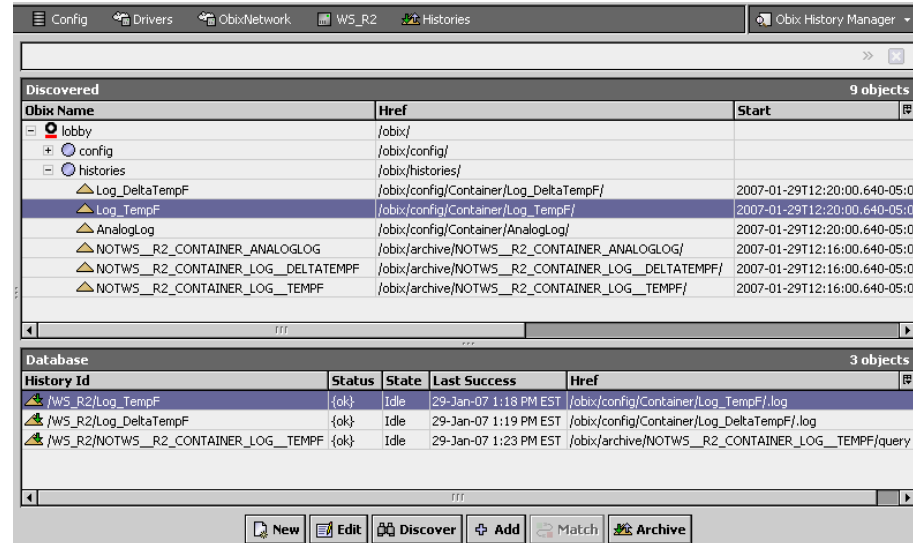
An R2ObixClient device's Watch Interval property is found on its **Points** extension, and defaults to 2 seconds.

- Typically on an R2ObixClient device the watch interval should be increased to 10 seconds or more, to reduce the load on the R2 platform.
- If the R2 station takes longer than 15 seconds to return the watch once polled, you may need to increase the session timeout. The R2ObixClient device has a sessionTimeout slot that is hidden by default. Go to the device's slot sheet and remove the hidden flag from the slot. Once visible, you can go to the property sheet of the R2ObixClient and adjust upward to tune, if needed.
- Enabling debug on the R2 server can be useful to diagnose initial problems, but should never be used unless necessary, as it *greatly* slows the server's response time to the Ax Client.
- If you encounter errors similar to:
HTTP Error 503:Service Unavailable
this indicates the R2 host has run out of webservice threads. See [“Other possible R2 host changes”](#) on page 2-6.

R2ObixClient Histories (logs and archives)

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

Figure 2-20 ObixHistoryManager is for importing R2 logs and/or archives into the history space



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

Each added descriptor (after archived in AX) produces one history in the station's local history space, organized by default under a container with the same name as the source R2ObixClient. Figure 2-21 shows an example of how the created histories appear in the AX station's history space.

Figure 2-21 By default, imported histories under container with name of ObixClient (often, stationName).

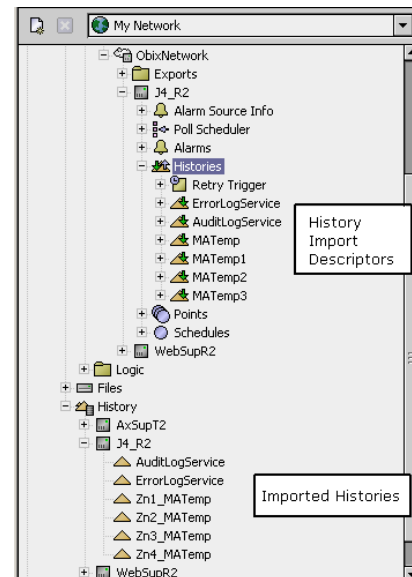


Figure 2-22 shows an example add/edit dialog for an ObixHistoryImport descriptor with default values, where the import descriptor name matches the source R2 Log object name, and the history ID is a combination of the R2ObixClient's name / Log object name.

Figure 2-22 Add/Edit dialog for importing an R2 log or archive by import descriptor

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

☐ Name: Log_TempF
☐ History Id: /WS_R2 / Log_TempF
☐ Execution Time: Daily Time Of Day: 02:00:00 AM EST Randomization: +00000h 00m 00s Days Of Week: ☒ Sun ☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☒ Sat
☐ Enabled: true
☐ Capacity: Unlimited
☐ Full Policy: Roll
☐ Href: /obix/config/Container/Log_TempF/.log

OK Cancel

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

ObixClient history import notes

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

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

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

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

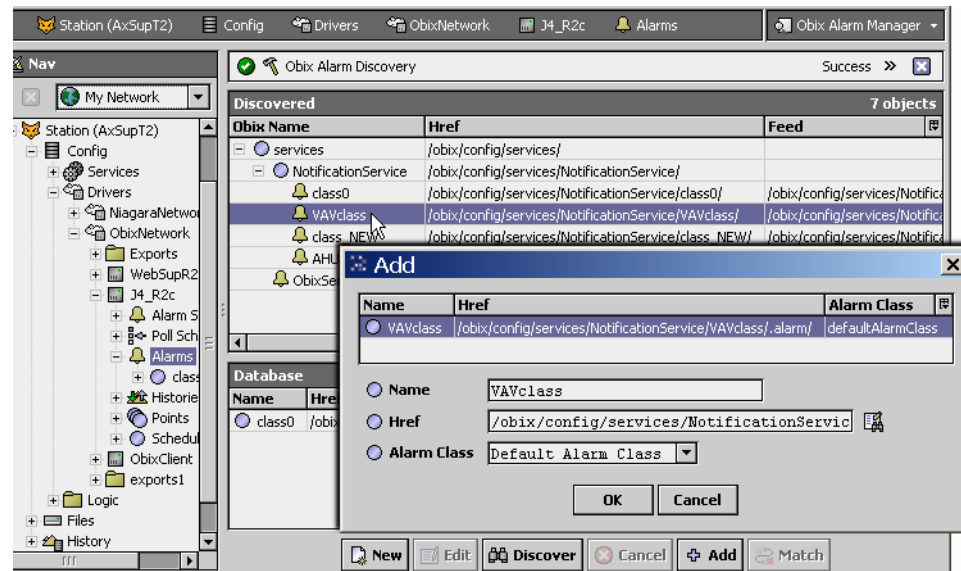
Although the Obix History Import manager allows you to create multiple import descriptors (with default values) for an identical Obix Name, note that only *one* can successfully import using the *same History Id*. Import descriptors with a duplicate History Id will go into fault upon import attempt.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

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

R2ObixClient Alarms

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

Figure 2-23 Obix Alarm Manager view used to add alarm feeds from R2 NotificationClass objects



To configure, double-click the R2ObixClient's **Alarms** extension, and in the Obix Alarm Manager view, perform a **Discover**. Expand the **services** node, and then the **NotificationService** node. As shown in Figure 2-23, each R2 NotificationClass object is represented as a separate alarm feed, in addition to a “global” “ObixService” alarm feed.

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

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

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

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

See the following subsections for additional R2 alarm import details:

- [R2 alarm import operation](#)
- [Example R2 alarm imports](#)
- [Final notes on imported R2 alarms](#)

R2 alarm import operation

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

Thus, as shown in Figure 2-24, all R2 native alarms appear as “normal” (green alarm bell) source state events when routed to an AX Alarm Console. However, all of the “metadata” about each event, including the alarm state (normal, offnormal, etc.), exceeded value, and so on, is included in the alarm details of the alarm record.

Figure 2-24 Imported R2 alarms always have “Normal” source state in Alarm Console

Open Alarm Sources

5 Sources / 17 Alarms

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

fff

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

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

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

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

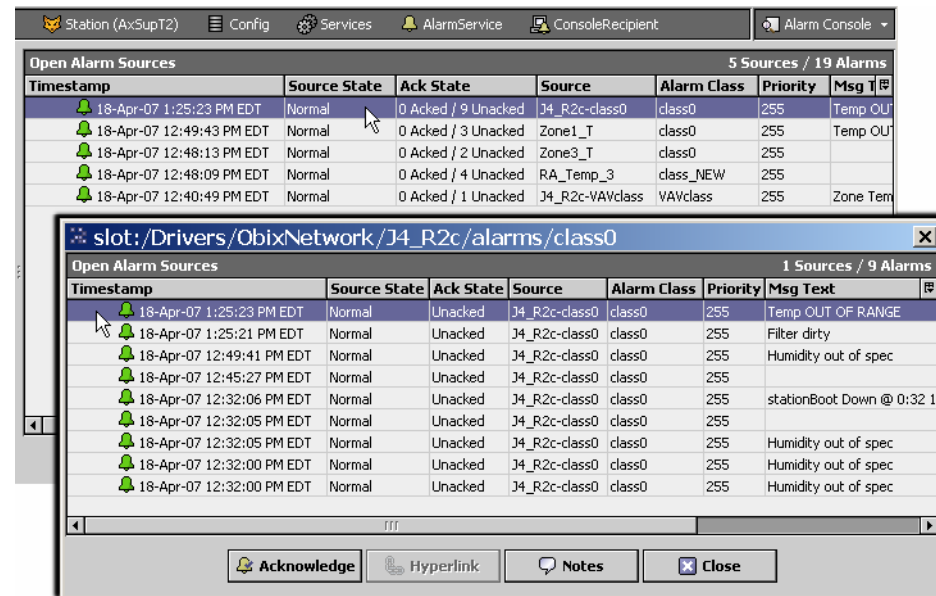
Note: *In this case especially, you must be careful about acknowledgement of a single row in the Alarm Console, as it may represent several different R2 alarm sources—the single acknowledgement will be applied to all underlying alarms (whether you are aware of them or not).*

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

Example R2 alarm imports

As previously noted, all imported R2 alarms and alerts from any R2 station appear as “normal” source state events. Also, it is possible that alarms from different R2 source objects will appear under a single row in the Alarm Console (if they are not mapped as Obix proxy points). If you double-click to investigate, you see that they also appear as “normal.” As needed, double-click rows again for the final Alarm Record dialog. See [Figure 2-25](#) and [Figure 2-26](#).

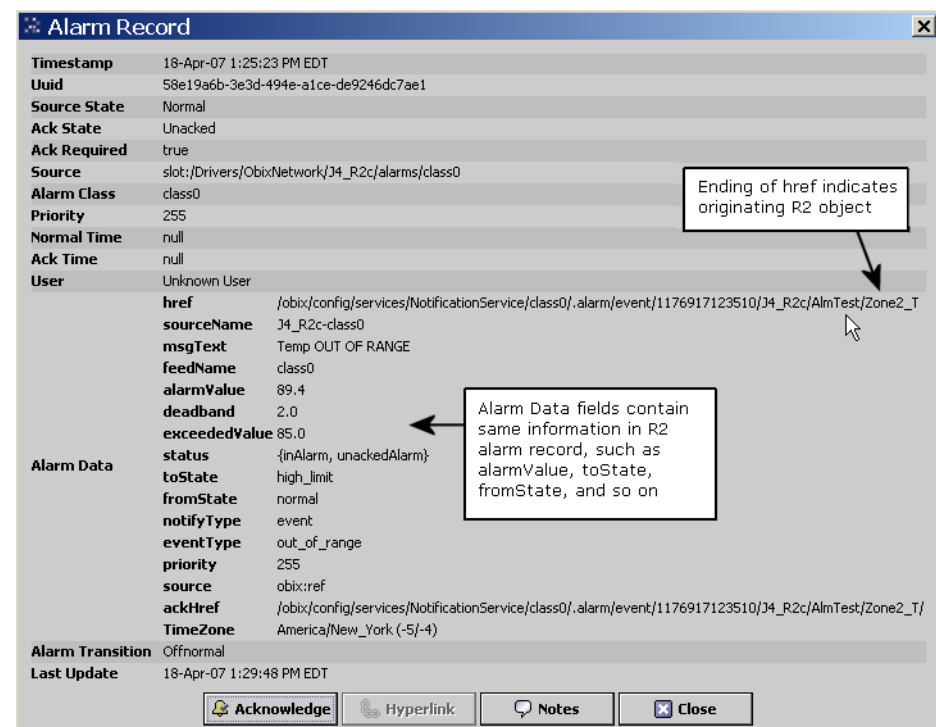
Figure 2-25 R2 alarms by objects not mapped as proxy points require Alarm Details inspection



Timestamp	Source State	Ack State	Source	Alarm Class	Priority	Msg T
18-Apr-07 1:25:23 PM EDT	Normal	0 Aced / 9 Unacked	J4_R2c-class0	class0	255	Temp OUT
18-Apr-07 12:49:43 PM EDT	Normal	0 Aced / 3 Unacked	Zone1_T	class0	255	Temp OUT
18-Apr-07 12:48:13 PM EDT	Normal	0 Aced / 2 Unacked	Zone3_T	class0	255	
18-Apr-07 12:48:09 PM EDT	Normal	0 Aced / 4 Unacked	RA_Temp_3	class_NEW	255	
18-Apr-07 12:40:49 PM EDT	Normal	0 Aced / 1 Unacked	J4_R2c-VAVclass	VAVclass	255	Zone Tem

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

Figure 2-26 Alarm Record details show source R2 object by path in href, along with R2 alarm data



Alarm Record	
Timestamp	18-Apr-07 1:25:23 PM EDT
Uuid	58e19a6b-3e3d-494e-a1ce-de9246dc7ae1
Source State	Normal
Ack State	Unacked
Ack Required	true
Source	slot:/Drivers/ObixNetwork/J4_R2c/alarms/class0
Alarm Class	class0
Priority	255
Normal Time	null
Ack Time	null
User	Unknown User
href	/obix/config/services/NotificationService/class0/.alarm/event/1176917123510/J4_R2c/AlmTest/Zone2_T
sourceName	J4_R2c-class0
msgText	Temp OUT OF RANGE
feedName	class0
alarmValue	89.4
deadband	2.0
exceededValue	85.0
status	{inAlarm, unackedAlarm}
toState	high_limit
fromState	normal
notifyType	event
eventType	out_of_range
priority	255
source	obix:ref
ackHref	/obix/config/services/NotificationService/class0/.alarm/event/1176917123510/J4_R2c/AlmTest/Zone2_T/
TimeZone	America/New_York (-5/-4)
Alarm Transition	Offnormal
Last Update	18-Apr-07 1:29:48 PM EDT

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

Final notes on imported R2 alarms

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

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

erated the alarm in order to acknowledge it, regardless of what permissions (if any) that user may have on any of the NotificationClass objects.

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

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

- archiveMode=archive_local_no_SQL
- alarmArchiveAddress, checkbox cleared

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

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

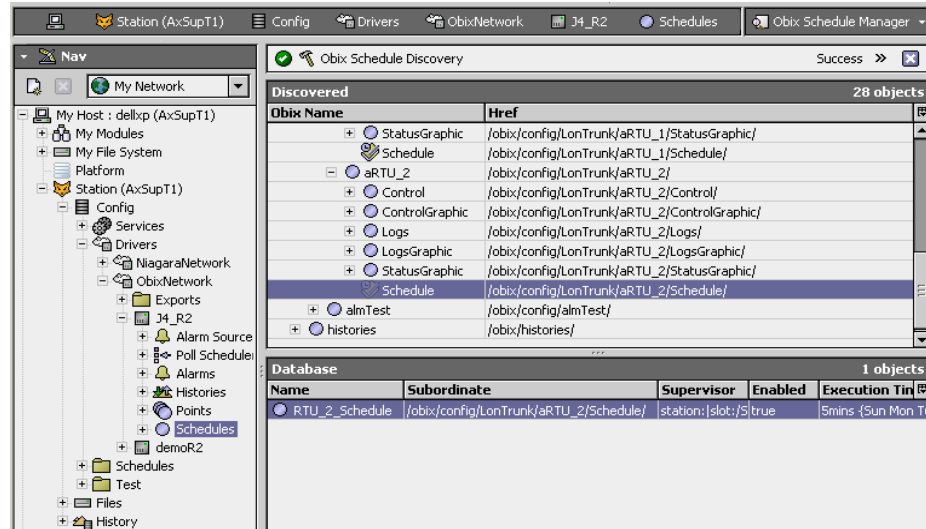
Alternative to importing native R2 alarms If alarming is critical, for the reasons previously noted you may wish to transition all alarming to “native AX” alarming, instead of using the ObixClient Alarms feature. Do this by creating Obix proxy points for all objects that require alarming (or runtime/COS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

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

R2ObixClient R2 Schedule exports

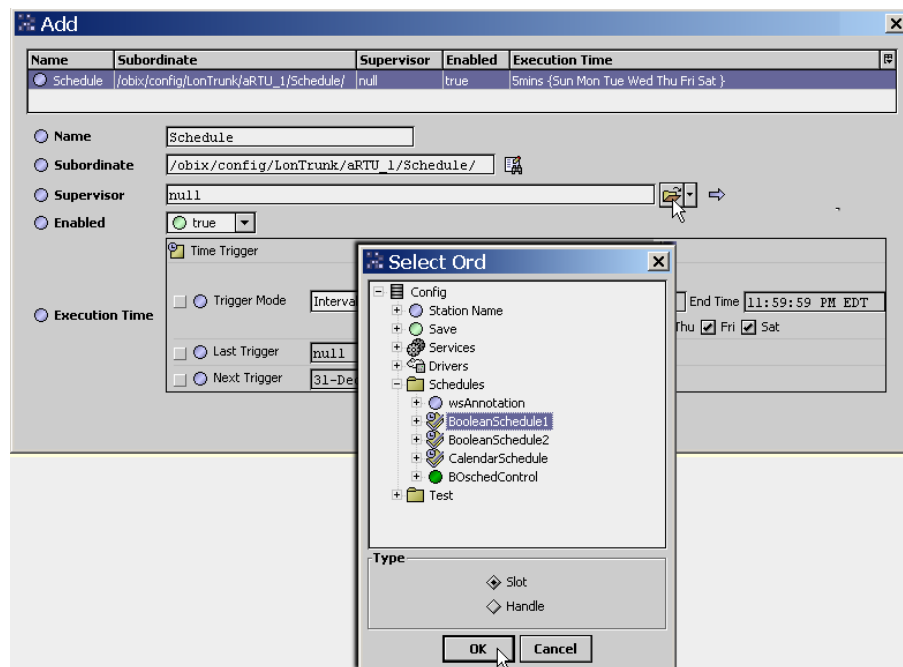
By default, each R2ObixClient has 4 device extensions: Alarms, Histories, Points and Schedules. Expand the R2ObixClient and double-click **Schedules** (R2ScheduleDeviceExt) for the Obix Schedule Manager view, and then perform a **Discover**. Expand the **config** branch of the **lobby** to locate target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

Figure 2-27 Obix Schedule Manager view of Schedules device extension to designate R2 slave schedules



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

Figure 2-28 Add/Edit dialog for ObixScheduleExport descriptor, showing Supervisor



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

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

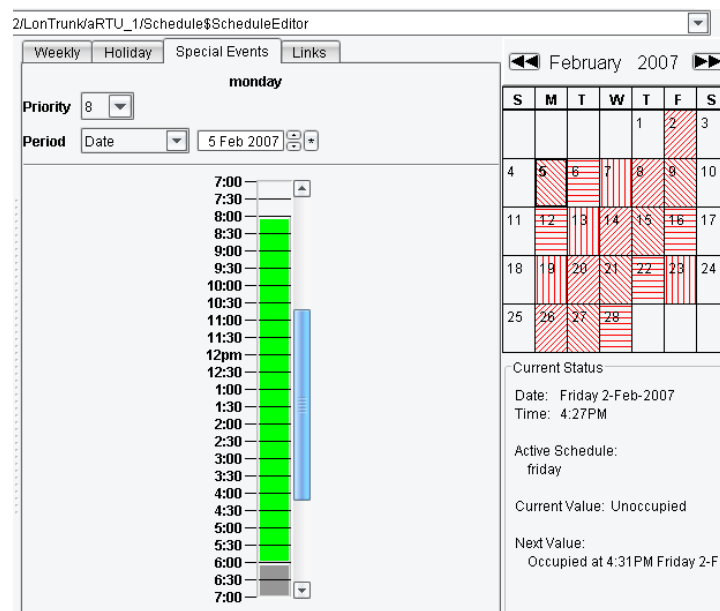
Note: Any target R2 Schedule should not already be “slaved” to an R2 Schedule in another station (typically in the R2 Web Supervisor station), otherwise the AX schedule supervisor function may not be successful. To remove a Schedule from R2 slaved control, delete the externalSubscription link on its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

AX Schedule to R2 Schedule operation

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

Note that unlike AX schedule special events (which “intermingle” with weekly events), R2 schedule special events *replace* the normal weekly schedule. As shown in [Figure 2-29](#), schedule events that are written to the R2 Schedule from the AX master appear in the “Special Events” tab of the JDE Schedule Editor view, in a block of around four weeks. Each has a “weekday” name (monday, tuesday, monday2, tuesday2, etc.) seen at the top of the tab.

Figure 2-29 R2 JDE Scheduler Editor view, Special Events tab, showing received AX schedule events



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

Document change log

Updates (changes/additions) to this *R2 to AX via oBIX - Engineering Notes* document are listed below.

- Updated: January 30, 2009
Reworked document to describe changes made in the both the R2 server and AX client oBIX software modules to increase proxy point polling efficiency. Now when making AX Obix proxy points, only *properties* of discovered R2 objects should be selected. See [“Specify the property”](#) on page 2-11. Other changes are in sections [“Requirements”](#), including new subsection [“Loading effects of oBIX integration”](#) on page 2-2, new sections [“Pre-integration R2 station changes”](#) on page 2-3, and [“Other possible R2 host changes”](#) on page 2-6. On the AX side, most changes are in sections [“About Discover”](#), [“Include” options](#), [“About Obix proxy points”](#), and [“ObixClient proxy point tips”](#). Other new sections related to proxy points are [“About forceUpdate”](#) on page 2-13, and [“Client polling timing”](#) on page 2-15.
- Updated: December 10, 2007
Applied new formatting for printed document.
- Publication: June 13, 2007
Minor changes related to “obixInternal.properties” file/feature for R2 station hosts, as mentioned in sections [“Requirements”](#) on page 2-2 and [“About Discover”](#) [“Include” options](#) on page 2-10, also re-

- flected in updated screen capture for [Figure 2-10](#) on page 9. Removed “Beta Draft” header and change bars in PDF.
- Draft: May 24, 2007
Third draft “Engineering Notes” type document. Various changes to document reflecting improvements in the build 3.1.31 and AX-3.2 versions of the obixDriver (coupled with changes to the R2-side obix module, 2.301.522f.beta). Blue change bars in the margins of the PDF version of this document indicate general areas of change, with most changes within subsections under the [“R2ObixClient Points”](#) section.
 - Draft: April 18, 2007
Second draft “Engineering Notes” type document. Various changes to document reflecting improvements in the build 3.1.30.6 version of the obixDriver. Change bars in the margins of the PDF version of this document indicate general areas of change.
 - Draft: February 6, 2007
Initial first draft “Engineering Notes” type document.