KODARO

Haystack Driver Guide

January 22, 2018
This documents usage of the Haystack driver on the Niagara N4 platform.

OVERVIEW

This document serves as a guide for using of the Haystack serial driver on the Niagara N4 platform. This driver is intended to integrate the N4 tagging model directly into the Haystack tagging model. It provides a mechanism for two-way communication of tags between an N4 station and a third-party Haystack client. See *Appendix A: Application Notes* for further details concerning these options.

System Compatibility

This driver integrates directly to the Niagara N4 tagging database using the Haystack Rest API as defined by http://project-haystack.org/.

Supported Ops

- About
- Ops
- Read
  - The read op deviates from the haystack protocol slightly by not following the haystack filter syntax. This is because Niagara does not strictly follow a haystack compliant database and rather uses NEQL as its tag querying system. The filter parameter should follow NEQL syntax for the filter parameter when using this op.
- Watch Sub
- Watch Unsub
- Watch Poll
- Point Write
- His Read
- Invoke Action
- Query
  - The query op is a non-standard Haystack operation added for supporting Niagara's native query languages, bql and neql. See Application notes for details on its usage (Using the Query Op).

Niagara Compatibility

This software will function on all Niagara N4 4.*n.nnn* platforms.

Workflow Summary

1. Module Installation
2. Adding the Driver Object
3. License Driver Object
4. Adding Network Objects

5. Configuring the Network
6. Adding Device Objects
7. Adding Points

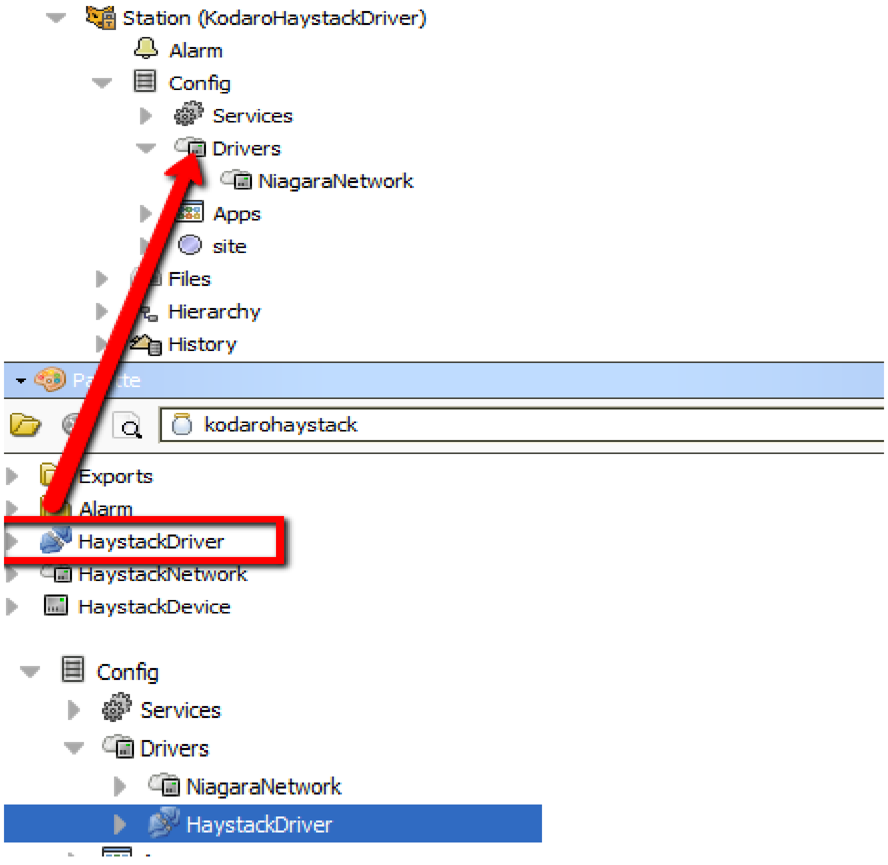
MODULE INSTALLATION

Install the driver module on the computer where Niagara N4 Workbench will be run. To install, place a copy of the file in the modules directory of your Niagara N4 installation. This is typically C:\Niagara\Niagara-4.n.nnn\modules. Restart station.

Install the module on the target station. Using a Niagara N4 Workbench where the module has already been installed, connect to the station's platform service. Go to the Software Manager and install. Restart station.

ADDING THE DRIVER OBJECT

Open the driver palette in the engineering tool. Copy and paste the HaystackDriver object under the Drivers node in the station database.

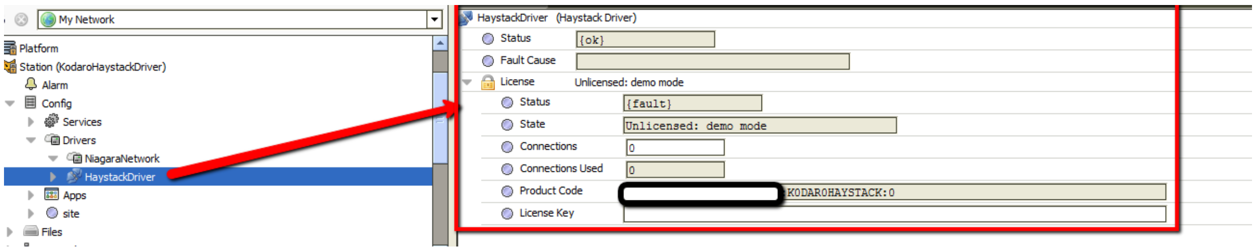


LICENSING THE DRIVER OBJECT

Licensing is based off connections. Each connection is defined as a connected device. This means any device added to any network will count against a connection whether that connection is active or not. The ability of the driver to receive haystack rest ops does not rely on connections. If it is licensed for any number of connections, this feature operates regardless of the amount incoming connections. The demo status of the driver will last for two hours at each station start. After demo mode expires, the station must be restarted to resume operation, but it is otherwise safe to build a database.

The licensing object is located on the property sheet of the HaystackDriver. It has the following properties.

- Product Code – Text automatically generated by the driver that is needed to generate a license key.
- License Key – Where the key to validate the license must be entered.
- Devices – The number of devices under all networks to license.



HaystackDriver (Haystack Driver)

Status	{ok}
Fault Cause	
License	Unlicensed: demo mode
Status	{fault}
State	Unlicensed: demo mode
Connections	0
Connections Used	0
Product Code	[redacted] KODAROHAYSTACK:0
License Key	

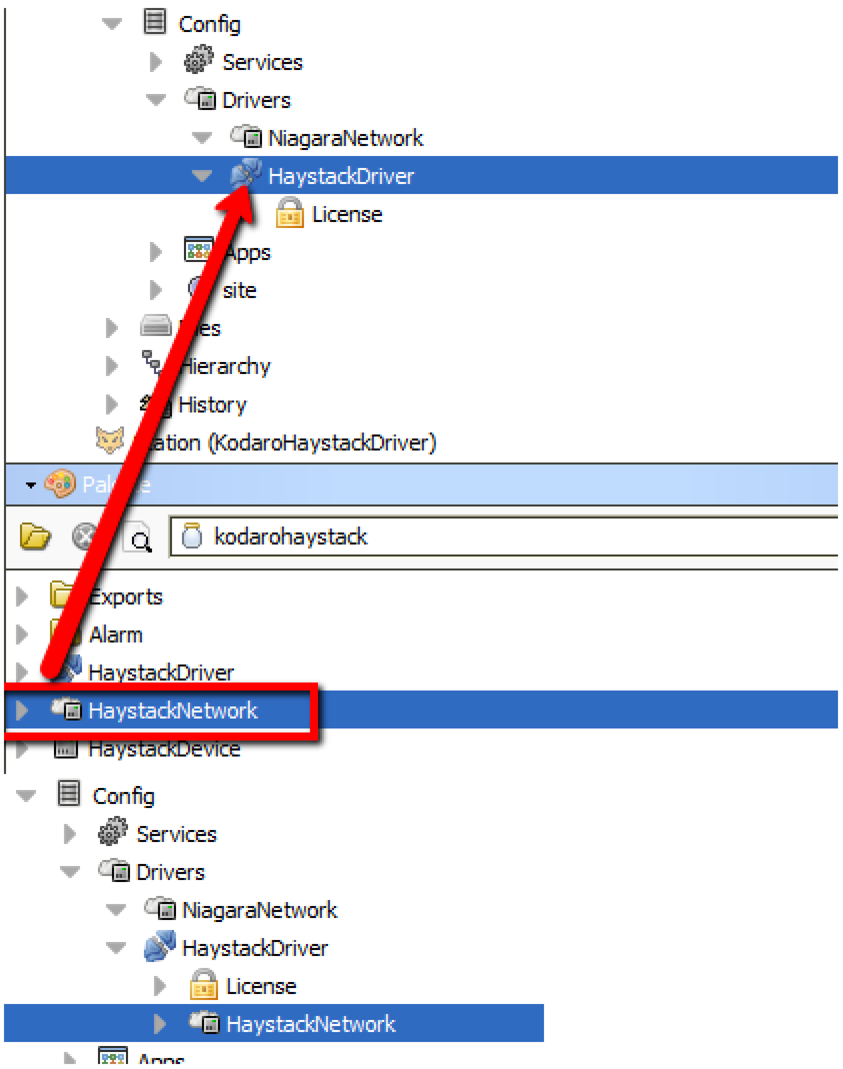
Set the number of devices you plan to use. Copy the value of the "Product Code" property that is automatically generated. You should highlight the value and copy it by pressing CTRL-C. Send the product code to your Kodaro representative. They will respond with a text string for you to enter in the "License Key" property.

You must restart the station after changing the "License Key".

The exact text and case of the product code and license key are case sensitive and critical to licensing. Please do not send screen shots. Highlight the text, copy it and paste into an email when requesting a license.

ADDING NETWORK OBJECTS

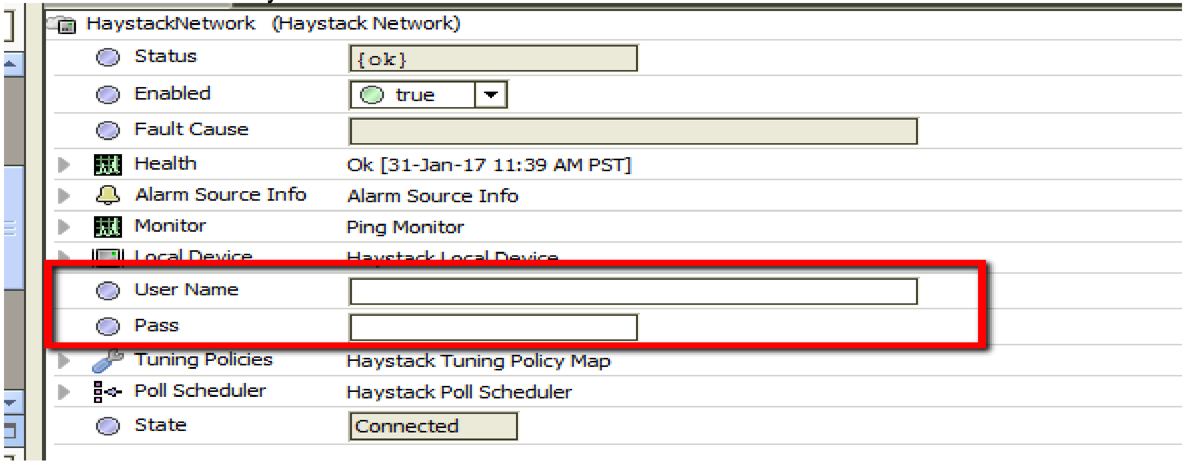
Open the driver palette in the engineering tool. Copy and paste the appropriate network object under the driver node in the station database.



CONFIGURING THE NETWORK

The driver uses the ping monitor to determine when to try to establish a connection. Wait for the driver to connect or manually invoke the ping action.

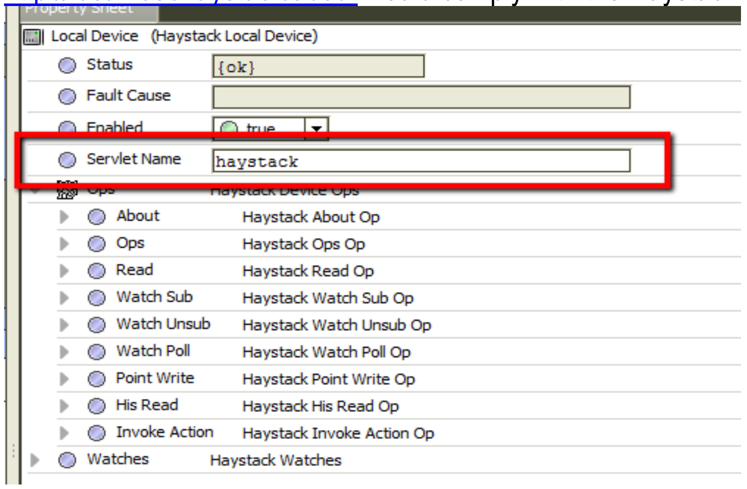
There are very few properties to configure to get a HaystackNetwork up and running. The only properties that may need to be configured are the User Name and Pass(password). If these are set, they will be used to authenticate all device connections. If these are not set, they can be set on each device individually and/or overridden on the device connection.



HaystackNetwork (Haystack Network)

Status	{ok}
Enabled	true
Fault Cause	
Health	Ok [31-Jan-17 11:39 AM PST]
Alarm Source Info	Alarm Source Info
Monitor	Ping Monitor
Local Device	Haystack Local Device
User Name	
Pass	
Tuning Policies	Haystack Tuning Policy Map
Poll Scheduler	Haystack Poll Scheduler
State	Connected

The Local Device object holds information about what Haystack Operations are available for a client to communicate with as well as a way to track point subscriptions handled for the watchSub/watchPoll ops. There is only one property that may need to be changed if desired, Servlet Name. Servlet name identifies the end point for all Rest API calls, i.e. http://localhost/haystack/about would comply with the Haystack About REST op about.



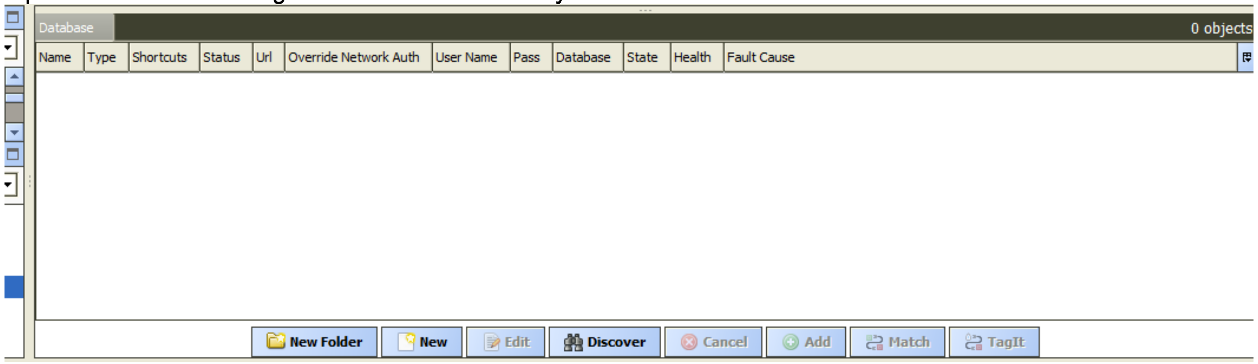
Local Device (Haystack Local Device)

Status	{ok}
Fault Cause	
Enabled	true
Servlet Name	haystack
Ops	Haystack Device Ops
About	Haystack About Op
Ops	Haystack Ops Op
Read	Haystack Read Op
Watch Sub	Haystack Watch Sub Op
Watch Unsub	Haystack Watch Unsub Op
Watch Poll	Haystack Watch Poll Op
Point Write	Haystack Point Write Op
His Read	Haystack His Read Op
Invoke Action	Haystack Invoke Action Op
Watches	Haystack Watches

You can also configure the amount of resources(threads) to be allocated to Watches. More detail on this topic can be found in *Appendix A: Application Notes*

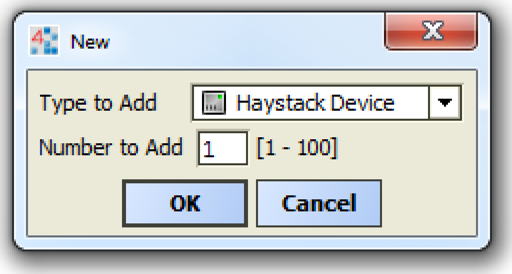
ADDING DEVICE OBJECTS

Open the *Device Manager* view of the Network you wish to add a device to.



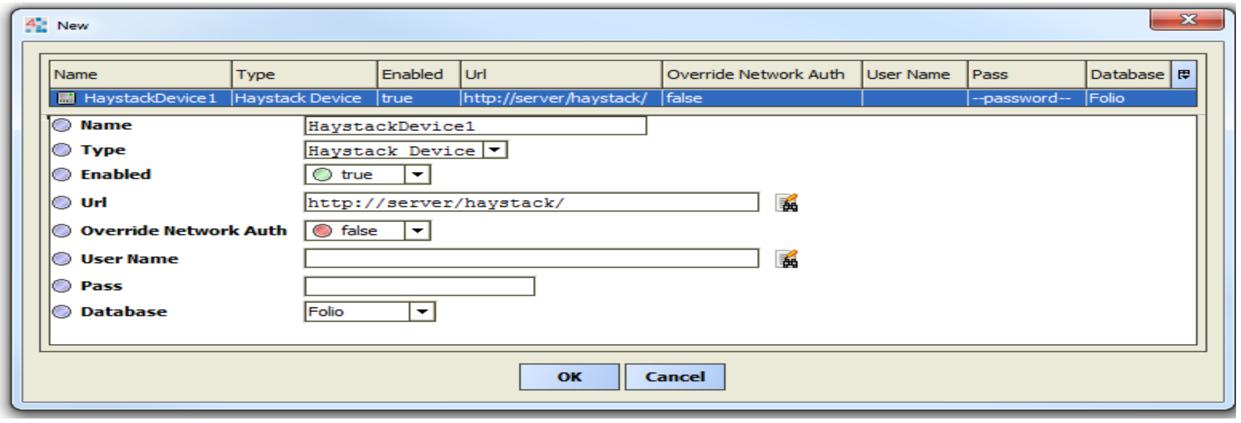
Name	Type	Shortcuts	Status	Url	Override Network Auth	User Name	Pass	Database	State	Health	Fault Cause

Use the *New* button to add a new Device. This will bring up a dialogue asking how many devices you wish to add.



Type to Add	Haystack Device
Number to Add	1 [1 - 100]

Enter the amount you wish and select Ok. This will bring up a another window where you can configure the devices.



Name	Type	Enabled	Url	Override Network Auth	User Name	Pass	Database
HaystackDevice1	Haystack Device	true	http://server/haystack/	false		--password--	Folio

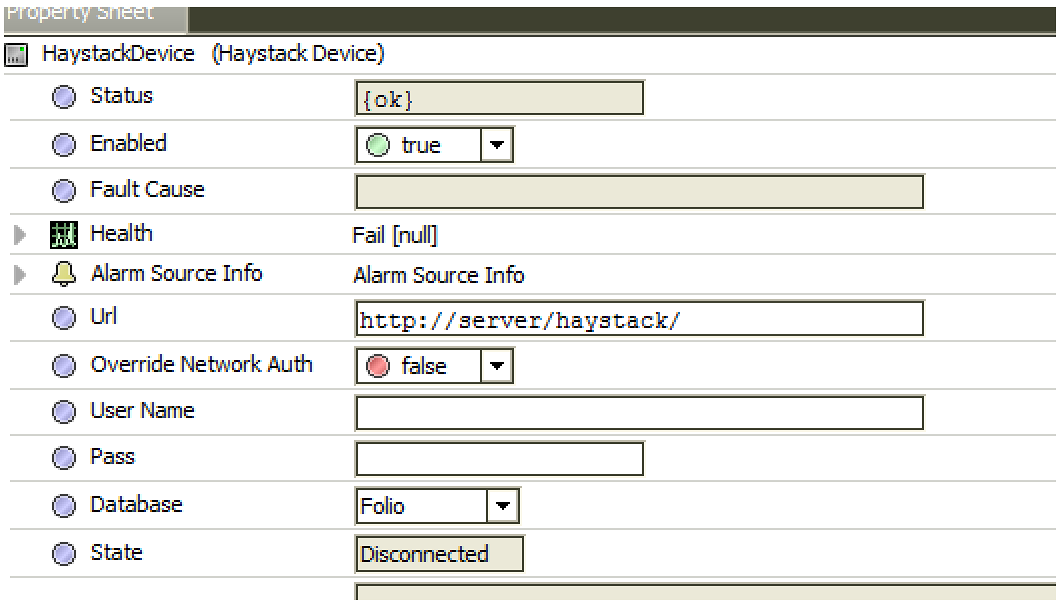
Name	HaystackDevice1
Type	Haystack Device
Enabled	true
Url	http://server/haystack/
Override Network Auth	false
User Name	
Pass	
Database	Folio

It is generally only necessary to set the Url assuming you have set authentication credentials at the network level. If network credentials have not been setup or are different from them, then it will be necessary to set the specific device authentication parameters at this time.

CONFIGURING THE DEVICE

It is generally only necessary to set the Url assuming you have set authentication credentials at the network level. If network credentials have not been setup or are different from them, then it will be necessary to set the specific device authentication parameters at this time.

a. Setting the URL – This must be a fully resolvable URL to the location of the server that accepts haystack requests. For a SkySpark(Folio) database, this format looks like: http://<ip address>/api/<project name>
b. Override Network Auth: This is false by default meaning it will use the User Name and password defined on the Network Object. If set to true, the User Name and Password defined on this object will be used instead.
c. User Name – User name to use during authentication if Override Network Auth is set to true.
d. Pass – Password to use during authentication if Override Network Auth is set to true.
e. Database – Folio and Unknown are the only two options. Folio is default as it is currently the only known haystack compliant database available.

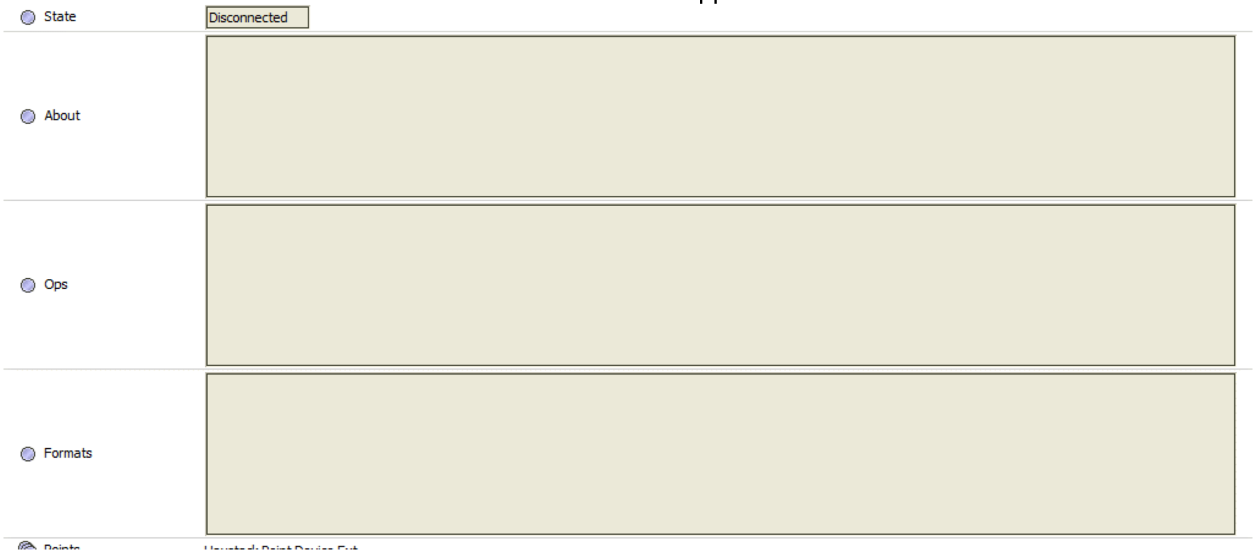

Property Sheet

HaystackDevice (Haystack Device)

Status	{ok}
Enabled	true
Fault Cause	
Health	Fail [null]
Alarm Source Info	Alarm Source Info
Url	http://server/haystack/
Override Network Auth	false
User Name	
Pass	
Database	Folio
State	Disconnected

CONNECTING TO A DEVICE

a. To make a connection, execute the connect or ping action on a device to make the connection. If this is successful, this will populate meta data about the haystack database that has been connected to.
   i. About – Shows basic data about project/database
   ii. Ops – Shows which operations are supported.
   iii. Formats – Shows which data formats are supported



COMPONENT REFERENCE

HaystackDriver

This is the root node of the driver. It represents a tree of Haystack networks.

Properties:

- **License** – See the section title “Licensing”.

HaystackNetwork

This represents a container for Haystack Devices. It is a network-level component in the NiagaraN4 architecture and has standard network component properties (see "Driver Architecture / Common network components" in the *NiagaraN4 User Guide* for more information).

The following properties are unique or have special importance:

- **User Name** – The user to authenticate connections with
- **Pass** – The password to authenticate connections with
- **Local Device** – Device responsible for listening to Haystack REST API calls.
  - **Servlet Name:** URL endpoint to this station to send all REST API calls. Format Example: http://<ipAddress/<Servlet Name>/<op name> (may also be https for secure connections).
  - **Ops –** Contains the Haystack REST ops supported
  - **Watches –** A new watch Object will be added each time a new watchSub is received. Every watch runs on its own thread.
    - WatchId : The reference used by the Haystack client to poll this watch.
    - Lease Time: How long to keep this watch alive in between polls.
    - Last Poll: The last time this watch was polled
    - Subscriptions: The amount of components this specific watch is currently subscribed to.

HaystackDevice

This is a device-level component in the NiagaraN4 architecture and has standard device properties (see "Driver Architecture / Common device components" in the *NiagaraN4 User Guide* for more information).

The following properties are unique or have special importance:

- **Url** – This is the fully qualified URL that points to the Haystack REST end point you wish to connect to.
- **Override Network Auth** – When set to true, this indicates to the device to use it User Name and Pass properties instead of the ones defined on the network.
- **User Name** – The user to authenticate connections with
- **Pass** – The password to authenticate connections with
- **Database** – A list of known Haystack compliant databases that may have some intricacies not specifically defined by the Haystack protocol but are handled by this driver.
  - **Folio** – The database name of SkySpark installations.
  - **Unknown** – Any Haystack database wished to be connected to but is not found in this list. No special care outside of the strict Haystack protocol will be followed when communicating with a device set as such.

HaystackProxyExt

The following properties are unique or have special importance:

- **Address** – This is the name used in a point log to poll the value.
- **Device Facets** – The facets are used to map values into Niagara. For Boolean and Enum points, it is important the trueText/falseText or enum range be set.

TROUBLESHOOTING

Problems in General

- Is the driver licensed? Has the demo mode expired?
- Turn on debug and watch the console. Anything obvious?
  - Debug property is on the network property sheet.
  - Watch the Application Director console.
- Performance issues
  - Look at the Message Queue on the network property sheet
    - Discovery will be on the urgent queue. Are there a lot of messages on the urgent queue?
    - Invoke the dump action on the message queue and look at the application director to see what is on the queue.
- Try to rule out the driver.
  - Does the problem exist in other software such as HyperTerminal?

Trouble Connecting

- Check license state.
- Check the SerialConfig object on the network.
  - Is the correct com port selected?
  - Is it in use by another driver?
  - Is the baud rate correct?
- Turn on debug
  - Anything printing on console?
    - Try connecting with HyperTerminal.
    - Usually a cable problem if manually wired.
  - Does the output look wacky?
    - Verify the baud rate is correct.

Read Problems

- What is the fault cause in the point manager or on the point Proxy Ext?
- Try increasing the timeout.
- Could be a performance problem, look at the message queue (see Problems in General)

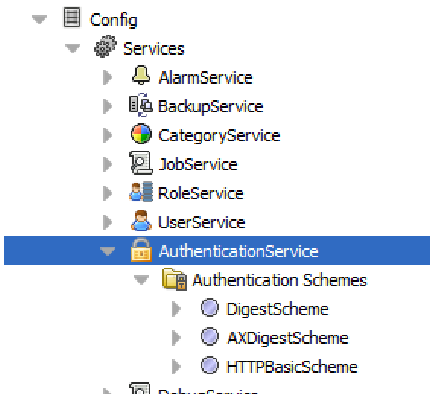
APPENDIX A: APPLICATION NOTES

Authentication Considerations

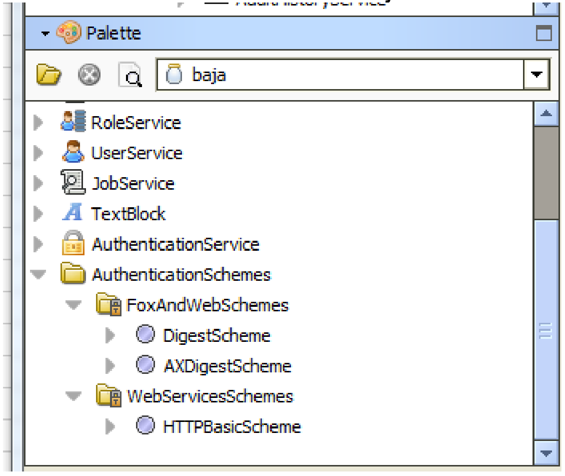
When authenticating between Niagara and SkySpark, there are a few items to consider.

1. What version of SkySpark are you running?
    a. 2.1.15 and below – This requires a Niagara user configured with the DigestScheme Authentication
    b. 3.0+ - This requires a Niagara user configured with the HttpBasicAuth Authentication.
2. Do I have the correct authentication scheme in my station?

    After determining which authentication scheme you need, you can check if you have it available by going to your AuthenticationService under the services container directly under config in your station.

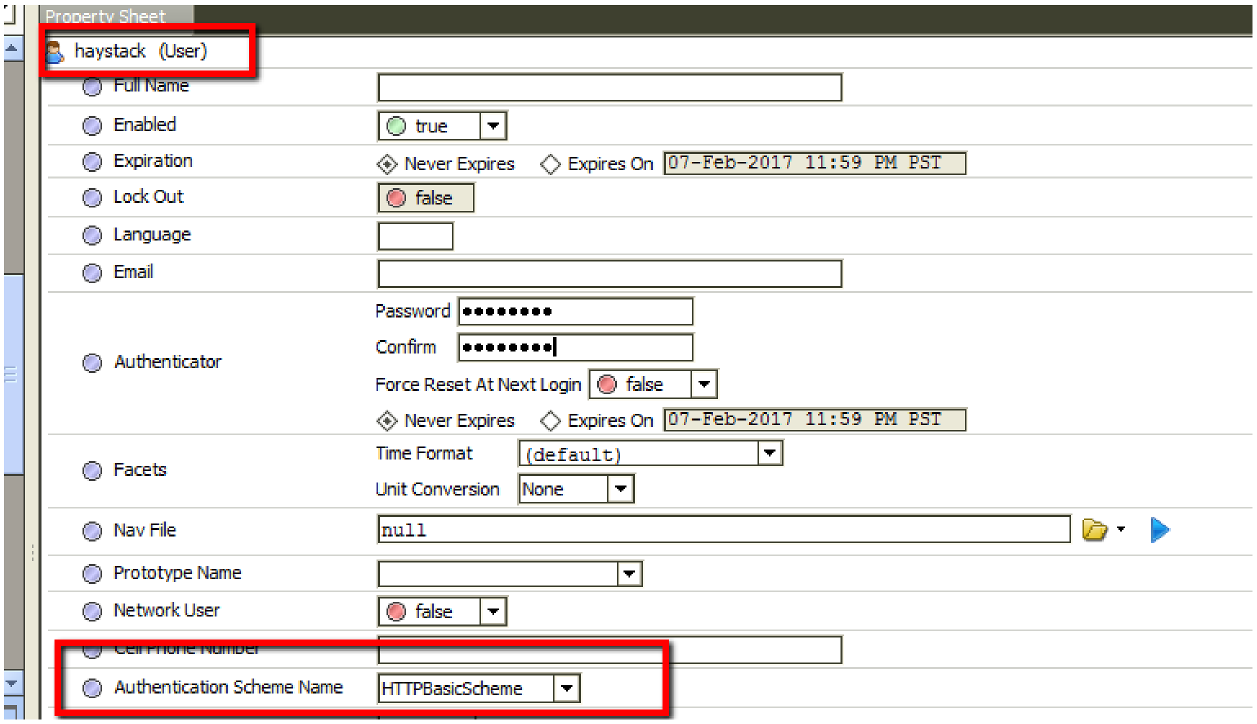


If you are missing an authentication scheme you need, you can find the built-in Niagara AuthenticationSchemes in the baja module under AuthenticationSchemes in the palette.



3. How do I configure a user to allow a connection into the station for haystack?

Assuming you have now installed the correct authentication schemes you need for the version of SkySpark you are using, you need to create a new user or configure an existing one to have this authentication scheme available for use. In this example, I have a user called haystack and assuming SkySpark 3.0+, I need to set the AuthenticationScheme to HttpBasicAuth. This can be done directly on the property sheet of the user.



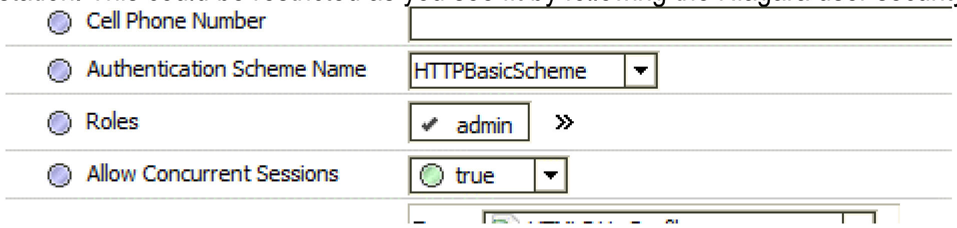
Property Sheet

haystack (User)

Full Name	
Enabled	true
Expiration	Never Expires / Expires On 07-Feb-2017 11:59 PM PST
Lock Out	false
Language	
Email	
Authenticator	Password ●●●●●●●●● Confirm ●●●●●●●●● Force Reset At Next Login false Never Expires / Expires On 07-Feb-2017 11:59 PM PST
Facets	Time Format (default) Unit Conversion None
Nav File	null
Prototype Name	
Network User	false
Cell Phone Number	
Authentication Scheme Name	HTTPBasicScheme

4. What about access restrictions?

The Haystack read and query op both respect the Niagara roles model. This means that even if I set my haystack user up with the correct Authentication Scheme but failed to give it role access, it would never allow logging in, or it might allow logging in but not able to discovery any points. By giving the user admin role access in this example, I’m assuming anybody that has this user info should have full access to the station. This could be restricted as you see fit by following the Niagara user security and role model.

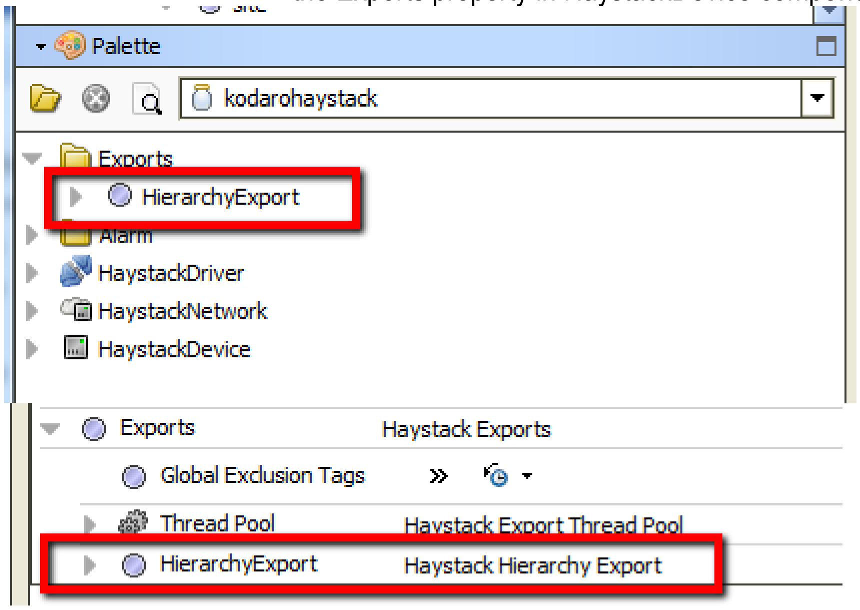


Cell Phone Number	
Authentication Scheme Name	HTTPBasicScheme
Roles	✓ admin
Allow Concurrent Sessions	true

Exporting the Niagara Database into a Haystack Database

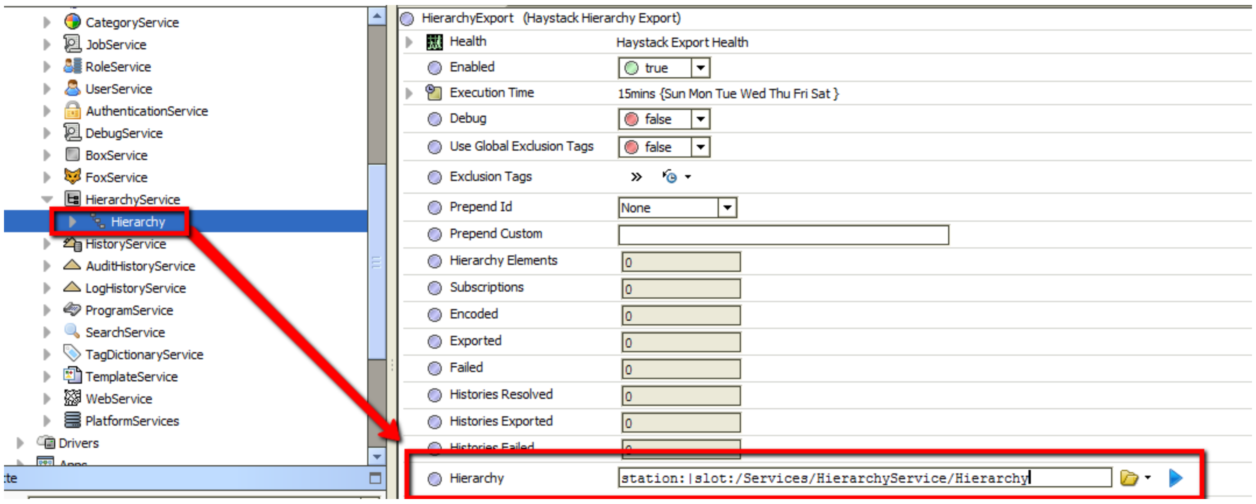
The export method of the Haystack Driver is designed to be unidirectional and push the data from Niagara into the Haystack Database. This is designed if you wish to keep all configurations of Haystack tagging done at the station.

1. Data is exported via a constructed hierarchy in the N4 station. Construction of these are beyond the scope of this document but a few restrictions to using these exist:
    a. Exporting with the use of anything other than QueryLevelDef and RelationLevelDef components when creating the hierarchy are not supported.
    b. When using a RelationLevelDev, Inbound must be true and RepeatRelation must be false.
    c. Open the kodarohaystack palette and drag a HierarchyExport component under the Exports property in HaystackDevice component added in step 5.



Exports	Haystack Exports
Global Exclusion Tags	
Thread Pool	Haystack Export Thread Pool
HierarchyExport	Haystack Hierarchy Export

d. Configuring the HierarchyExport
1. Reference the Hierarchy from the HierarchyService that you wish to export



- CategoryService
- JobService
- RoleService
- UserService
- AuthenticationService
- DebugService
- BoxService
- FoxService
- HierarchyService
  - Hierarchy
- HistoryService
- AuditHistoryService
- LogHistoryService
- ProgramService
- SearchService
- TagDictionaryService
- TemplateService
- WebService
- PlatformServices
- Drivers

HierarchyExport (Haystack Hierarchy Export)

Health	Haystack Export Health
Enabled	true
Execution Time	15mins {Sun Mon Tue Wed Thu Fri Sat }
Debug	false
Use Global Exclusion Tags	false
Exclusion Tags	»
Prepend Id	None
Prepend Custom	
Hierarchy Elements	0
Subscriptions	0
Encoded	0
Exported	0
Failed	0
Histories Resolved	0
Histories Exported	0
Histories Failed	0
Hierarchy	station:|slot:/Services/HierarchyService/Hierarchy

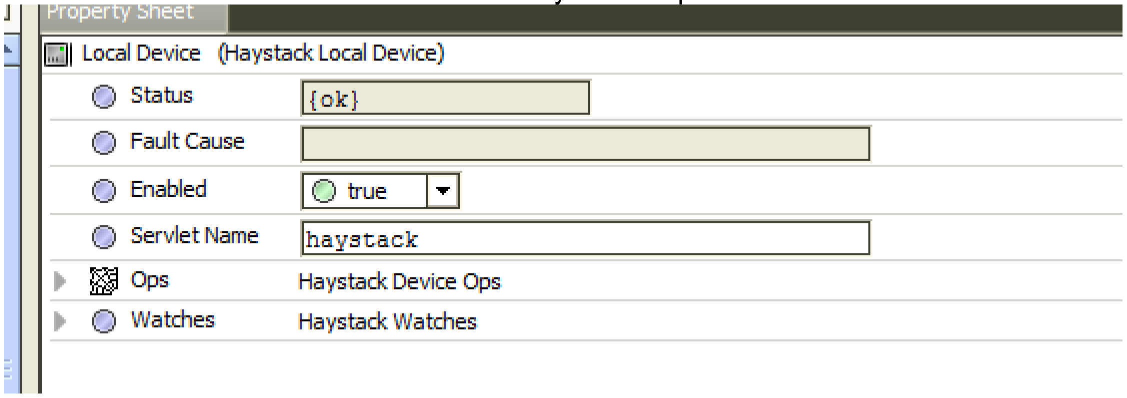
2. Inherit Relations – Defaulted to true. Inherits relationships from the hierarchy "parent" relationships. This is needed since N4 does not hold full relationship tree from every parent. For example, a point referencing (related to) a piece of equipment will have no idea what site that equipment is for but SkySpark/Folio needs to have this defined. By setting this to true, you will create an export of data that contains this level of information.
3. Export Histories – Defaulted to true. Define whether or not to export histories found on the components defined in the hierarchy.
4. Use Time Zone – Defaulted to False. Allows a complete override of the local station time zone for all time zone instances for historical data during the export.
5. Execution Time – Standard Niagara trigger that can be configured for Interval, Daily or Manul exports.

Using SkySpark to Integrate to a Niagara Station

The Haystack Driver can act as a Haystack server which can be used by any third-party Haystack client to integrate to. The following section describes this process as it pertains to the SkySpark platform.

For this method to work, you must make sure that the LocalDevice found on the HaystackNetwork is enabled and in an "ok" state. Take note of the servletName, this will be where you point the SkySpark Haystack connector. The example below assumes the default value of "haystack" and that the SkySpark instance and Niagara station are running on the same system and uses LocalHost in lieu of a specified IP address.

1. Make sure that the LocalDevice is ready. This requires a valid servletName and the Status is ok.

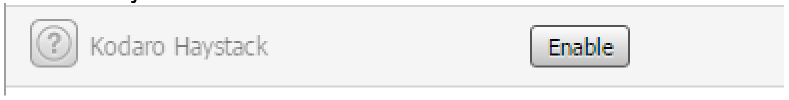


Local Device (Haystack Local Device)

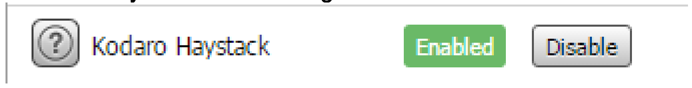
Status	{ok}
Fault Cause	
Enabled	true
Servlet Name	haystack
Ops	Haystack Device Ops
Watches	Haystack Watches

2. Install the kodaroHaystackExt.pod into SkySpark. This is included in the driver download. This is an extension pod that has some pre-built axon methods that will help to get a Niagara Station quickly integrated into a SkySpark installation. These methods serve as a guide to show how to get data from Niagara into SkySpark and how to modify data from SkySpark back to Niagara. These functions do not attempt to satisfy all possibilities but rather give a basic understanding on the process of using the Haystack Driver (Niagara) and Haystack Connector (SkySpark) to integrate a Niagara database into SkySpark. Under many circumstances these functions should be modified to meet the unique needs of your project. The extension pod does not need to be installed for the Kodaro Haystack Driver to work.
3. Once you have installed the kodaroHaystackExt.pod, you will find a Kodaro Haystack extension now available in the Extensions list in SkySpark. By default, this is disabled. Once enabled, this will make the example axon code available for use.

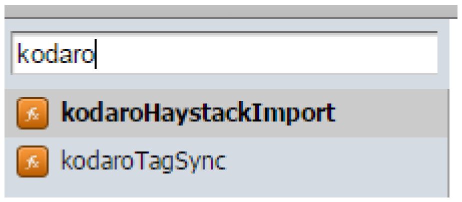
   Kodaro Haystack when first installed:



   Kodaro Haystack after being enabled:



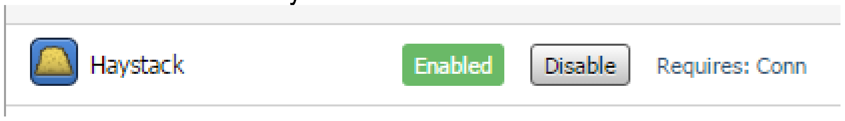
4. Once the Kodaro Haystack Extensions is enabled, you can verify the example scripts are available in the Func App. By typing kodaro in the search bar, you will get the list of kodaro scripts from this extension pod as they both start with kodaro. This search is performed from the Docs tab.



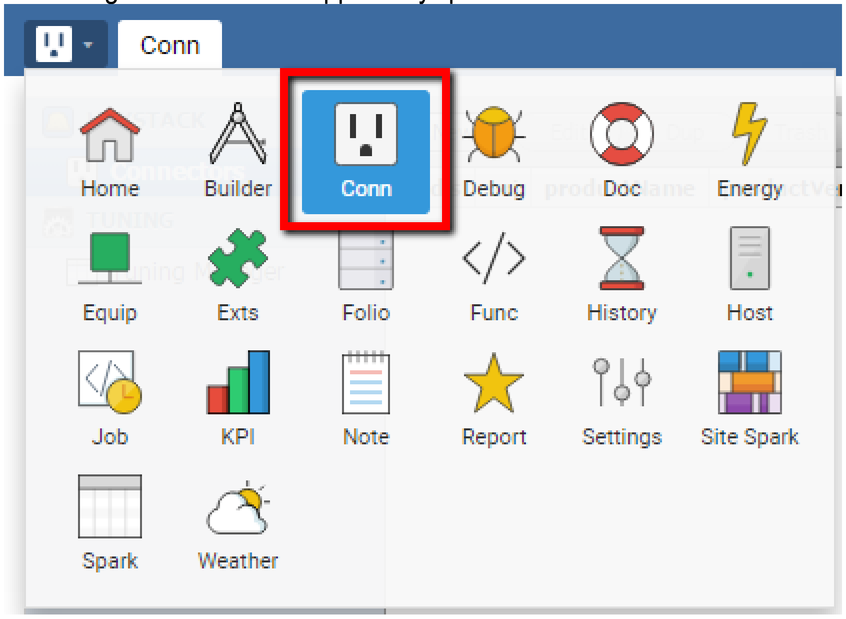
5. By clicking on any of these axon scripts, you will receive the full source code as well as a description of it's function. The latest information and source code is maintained in the scripts themselves so should always be referred to for the latest information rather than in this document.

   At this point it is assumed you have installed the kodaroHaystackExt.pod into SkySpark, enabled it and have a running Niagara N4 station with the Kodaro Haystack Driver with a Local Device ready to receive Haystack Rest Ops.

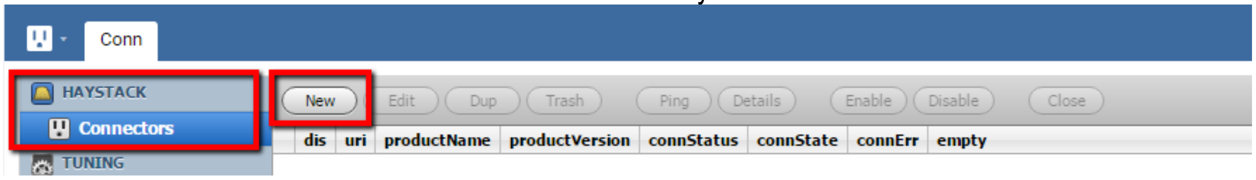
6. With everything installed on SkySpark, it is time to setup the Haystack Connector. See https://skyfoundry.com/doc/docTraining/HaystackConn for full details on this.
   a. Make sure that the Haystack Extension is enabled



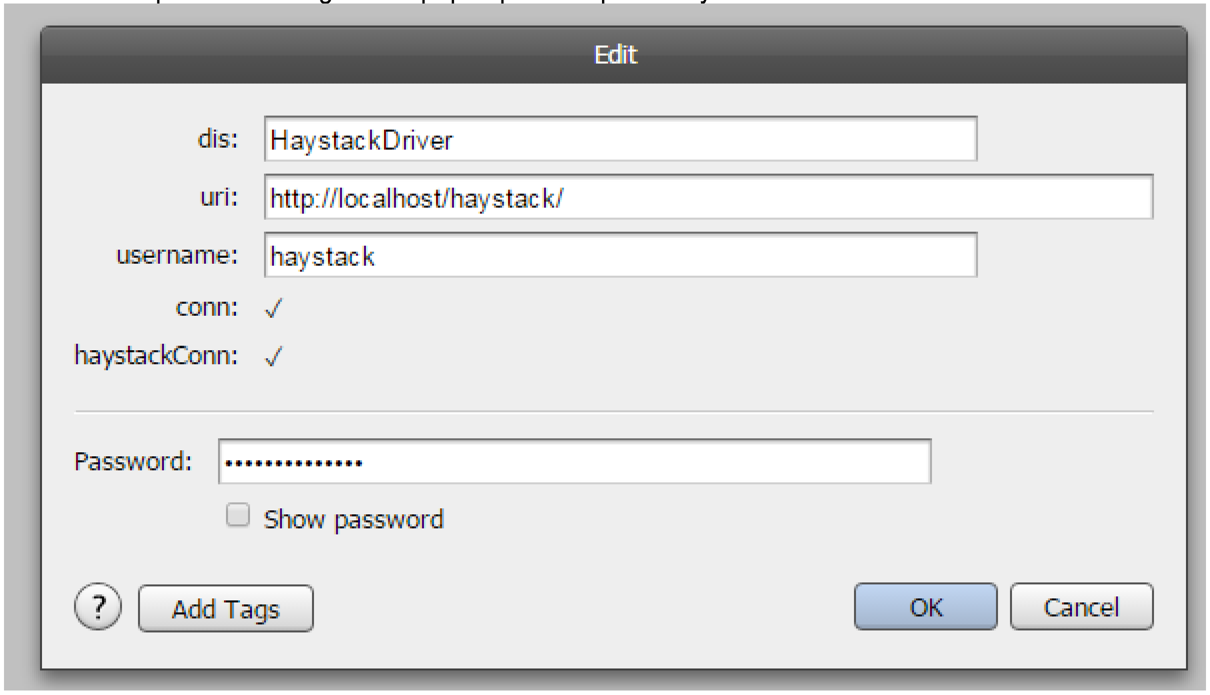
   b. Navigate to the Conn App in SkySpark



c. Select the "New" button to create a new Haystack Connector.



d. Complete the dialogue that pops up to setup the Haystack Connector.



Edit

dis: HaystackDriver

uri: http://localhost/haystack/

username: haystack

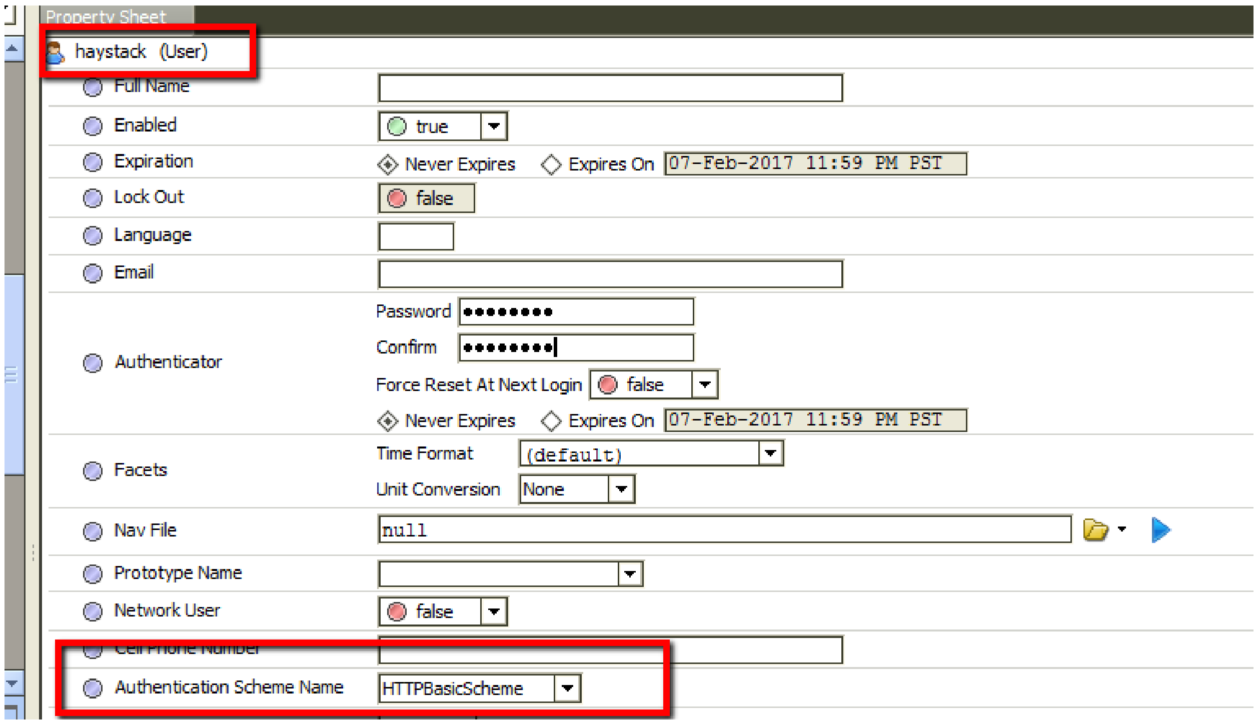
conn: ✓

haystackConn: ✓

Password: ••••••••••••••

Show password

i. dis – The display name of this connector. Multiple connectors can be created so naming this may be necessary when handling a system requiring more than one.
ii. uri - In this example, as stated before, is running on the same computer that is running the Niagara driver so we set the uri to point to http://localhost/haystack, haystack being the servletName configured in the LocalDevice of the driver and localhost referencing itself as the IP.
iii. username – This must be a user that exists in the Niagara N4 station that has been configured for HttpBasicScheme (for SkySpark 3.0.X) or DigestScheme (for SkySpark 2.1.X) in it's Authentication SchemeName.



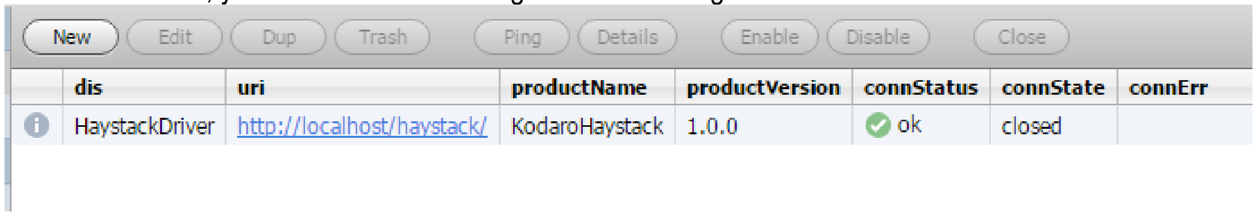
Property Sheet

haystack (User)

Full Name	
Enabled	true
Expiration	Never Expires / Expires On 07-Feb-2017 11:59 PM PST
Lock Out	false
Language	
Email	
Authenticator	Password ●●●●●●●●● Confirm ●●●●●●●●● Force Reset At Next Login false; Never Expires / Expires On 07-Feb-2017 11:59 PM PST
Facets	Time Format (default) Unit Conversion None
Nav File	null
Prototype Name	
Network User	false
Cell Phone Number	
Authentication Scheme Name	HTTPBasicScheme

iv. Password – Password for the Niagara user.

e. Once the Haystack Connector has been created you should see this in the connector table. You can manually invoke the ping method if it doesn't connect automatically. Once it connects, you should see something like the following:

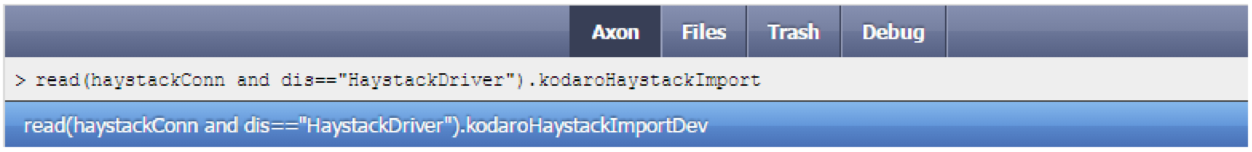


dis	uri	productName	productVersion	connStatus	connState	connErr
HaystackDriver	http://localhost/haystack/	KodaroHaystack	1.0.0	✓ ok	closed	

7. Now that we have connected SkySpark to Niagara, it's time to get data into SkySpark. We can accomplish this by examining the kodaroHaystackImport axon script found in the kodaroHaystackExt.pod. Since the full contents of the axon scripts are documented in the scripts themselves, the intention of this section is to describe its usefulness in an application context rather than trying to explain how it functions.
   a. What does the script do?
      The script can be broken down into 3 steps.
      1. Queries for all components in the Niagara station with the *hs:site* tag. All sites that are found are added into SkySpark.
      2. Queries for all components in the Niagara station with the *hs:equip* tag. All equipment that is found are added into SkySpark.
      3. Queries for all components in the Niagara station with the *hs:point* tag. All points that are found are added into SkySpark.

During the query process, the Haystack Read Op when handled in the Niagara Haystack Driver acts much like the Hierarchy Exporter when Inherit Relations is true. This means that if there is a relationship on a component, it is followed and continues up the relationship path to build a direct reference in the SkySpark database. This means a point with an equipRef but no siteRef would inherit a siteRef from the equipment it was pointing to, if that equipment had a siteRef.

b. How do you call the script?
The script takes a Haystack Connector as it's one parameter. This is the connector that's used to query Haystack Ops to complete the import.
Example:

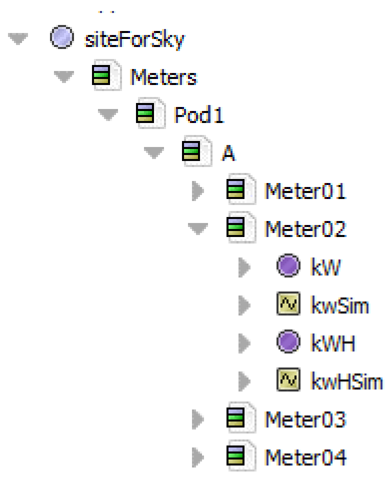


```
> read(haystackConn and dis=="HaystackDriver").kodaroHaystackImport
read(haystackConn and dis=="HaystackDriver").kodaroHaystackImportDev
```

The Axon code here first queries for Haystack Connectors (haystackConn) that are named "HaystackDriver" (dis). It then calls the kodaroHaystackImport function and passes in that result as the connector used. The result is a table of all the points added during the process. The sites and equips are not shown in the result but can be easily queried with Axon.

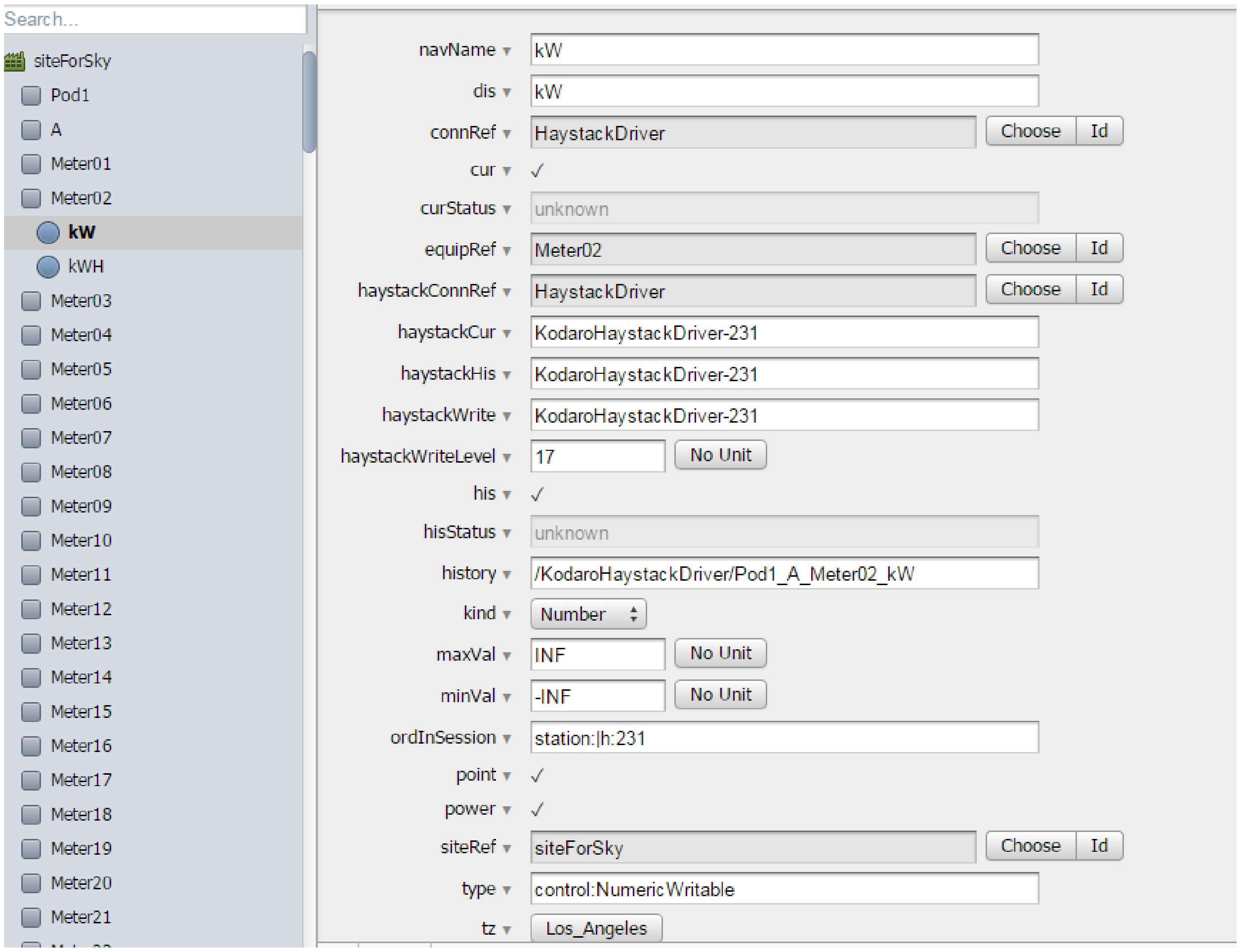
The script is designed to be only run once since it uses the add commit option. This means that if you need to run it again, the existing records will need to be deleted from SkySpark to prevent a duplicate record error. The script can be modified by the user to account for duplicates and update but is not shown in this document how to accomplish this.

c. What did we just do?
Let's take a look at what the database looks like in the Niagara station.



The data tree here shows the relationship to all the points and folders. What you don't see here is the Niagara Relations. All the points relate to the Meter via an equipRef and every folder relates to it's parent folder with an additional equipRef. Meters is the only folder that relates to the site (siteForSky) with a siteRef.

And now in SkySpark Builder App.



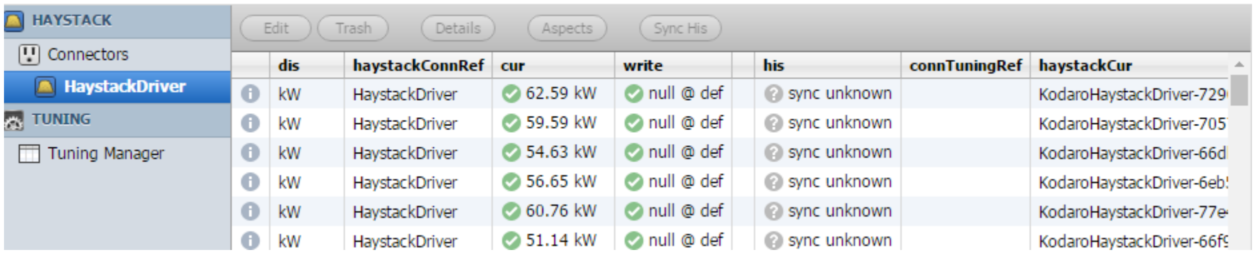
Search...

- siteForSky
  - Pod1
  - A
  - Meter01
  - Meter02
    - **kW**
    - kWH
  - Meter03
  - Meter04
  - Meter05
  - Meter06
  - Meter07
  - Meter08
  - Meter09
  - Meter10
  - Meter11
  - Meter12
  - Meter13
  - Meter14
  - Meter15
  - Meter16
  - Meter17
  - Meter18
  - Meter19
  - Meter20
  - Meter21

navName	kW
dis	kW
connRef	HaystackDriver
cur	✓
curStatus	unknown
equipRef	Meter02
haystackConnRef	HaystackDriver
haystackCur	KodaroHaystackDriver-231
haystackHis	KodaroHaystackDriver-231
haystackWrite	KodaroHaystackDriver-231
haystackWriteLevel	17 No Unit
his	✓
hisStatus	unknown
history	/KodaroHaystackDriver/Pod1_A_Meter02_kW
kind	Number
maxVal	INF No Unit
minVal	-INF No Unit
ordInSession	station:|h:231
point	✓
power	✓
siteRef	siteForSky
type	control:NumericWritable
tz	Los_Angeles

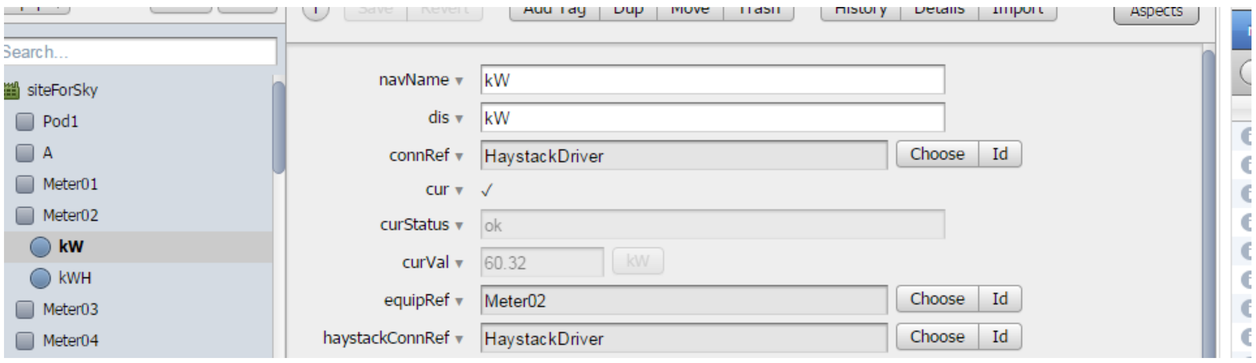
You can see the points are related to their equipment and have also inherited the relationships up the tree all the way to the siteRef even though the point in Niagara doesn't know about this.

a. What is missing? If you look closely at the screen shot above, you'll see the cur value is missing even though it is indicated to have one in Niagara. This is because a watch has not been created and polled against. This is done by setting up the tuning manager in SkySpark or by simply clicking on the Haystack Connector which will attempt to read all the points you just discovered via the import script.



dis	haystackConnRef	cur	write	his	connTuningRef	haystackCur
kW	HaystackDriver	✓ 62.59 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-729
kW	HaystackDriver	✓ 59.59 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-705
kW	HaystackDriver	✓ 54.63 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-66d
kW	HaystackDriver	✓ 56.65 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-6eb
kW	HaystackDriver	✓ 60.76 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-77e
kW	HaystackDriver	✓ 51.14 kW	✓ null @ def	sync unknown		KodaroHaystackDriver-66f

Back in the builder you can see that the cur values have now come in by simply looking at the points in the connector.

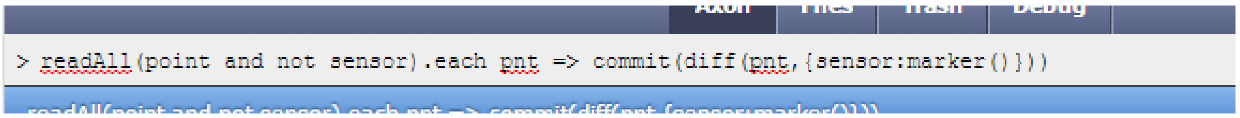


navName	kW
dis	kW
connRef	HaystackDriver
cur	✓
curStatus	ok
curVal	60.32 kW
equipRef	Meter02
haystackConnRef	HaystackDriver

As mentioned before, you will need to setup a SkySpark job to perform this or configure the tuning manager. Both of these topics are outside of the scope of this document.

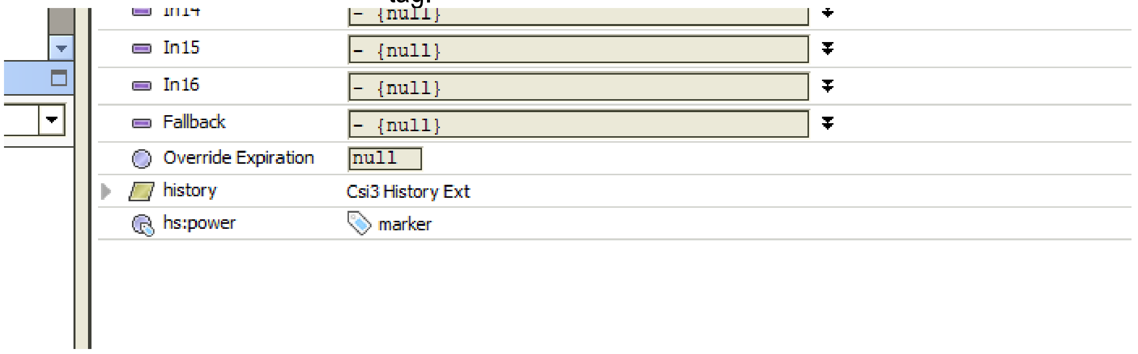
8. How do you make tag changes but keep the two databases in sync?
   i. Niagara To SkySpark – There is no prebuilt script or automatic process to identify tag changes Niagara and update SkySpark. However, the kodaroReadTags script will update SkySpark tags from the Niagara database. The script arguments include a grid of SkySpark points which the user would like to update and the Haystack Connector display name (dis).
   ii. SkySpark to Niagara – The kodaroWriteTags will push tags down to your Niagara database. In the example provided, you will find that no sensor/cmd/sp tag error exists on all the point records. This is because it is a SkySpark dependency and so a Niagara integrator may not remember to throw these on. In our example, we also know that every point we brought in is a sensor so we will now add sensor to every point and then run the kodaroWriteTags option to push these updates back down to Niagara.

      1. We first add a sensor tag to every point that doesn't have one using Axon.



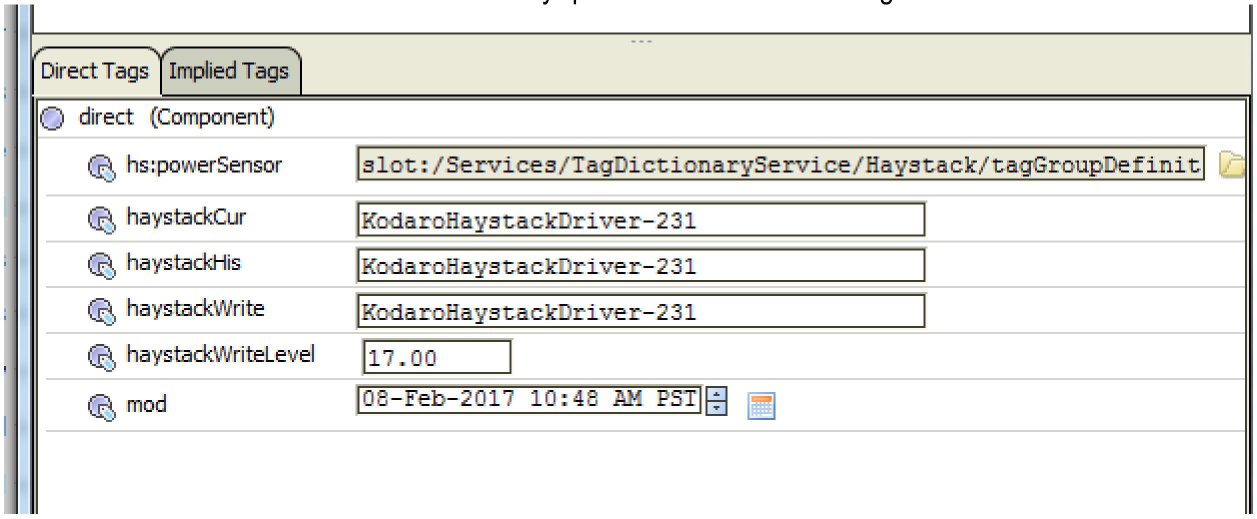
```
> readAll(point and not sensor).each pnt => commit(diff(pnt,{sensor:marker()}))
```

      2. Now we want to make sure Niagara knows about these changes so we send all the points back down to Niagara. Right now, there is no sensor tag.



In14	- {null}
In15	- {null}
In16	- {null}
Fallback	- {null}
Override Expiration	null
history	Csi3 History Ext
hs:power	marker

3. We'll run the kodaroWriteTags function for all points in SkySpark. After that, you see more than just sensor coming down but also any tags created in SkySpark that didn't exist in Niagara.

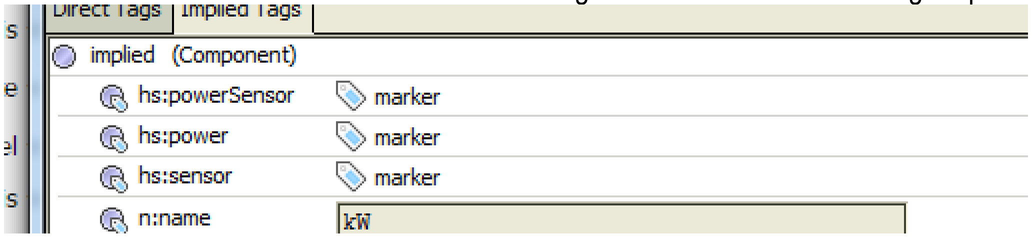


Direct Tags | Implied Tags

direct (Component)

hs:powerSensor	slot:/Services/TagDictionaryService/Haystack/tagGroupDefinit
haystackCur	KodaroHaystackDriver-231
haystackHis	KodaroHaystackDriver-231
haystackWrite	KodaroHaystackDriver-231
haystackWriteLevel	17.00
mod	08-Feb-2017 10:48 AM PST

But where is the sensor tag we added? You can see Niagara has created a tag group (hs:powerSensor). If we look at implied tags, we can see that the hs:sensor tag is now available on the Niagara point.



Direct Tags | Implied Tags

implied (Component)

hs:powerSensor	marker
hs:power	marker
hs:sensor	marker
n:name	kW

Using the Query Op

The non-standard haystack "query" op is designed to expand on the standard haystack query op but targeted to utilize the built-in query languages of Niagara (NEQL and BQL).

The query op follows the same format as the read op (http://project-haystack.org/doc/Ops#read) with a few additional parameters and modifications.

1. The filter parameter as defined by the read op in haystack is no longer haystack compliant but will need to conform to a proper BQL or NEQL query.

2. Additional Parameters to this query are "base" and "root".

    a. base – The base query language, this is either neql or bql.

    b. root – The root ord to originate the query from.

Example1: Considering the supporting SkySpark pod's documentation, you will find most of the examples using the read op. This is because it is designed to work as an example for any station. Here we will look at a more targeted query example knowing something about the station.

Let's assume our station has a very basic structure of:

Config

-HVAC

-Power

And I want SkySpark to run analytics on power only. This means that I only care about querying data out of the Power folder. Now I can try to do this with proper tag filtering and reading the whole database with the read op but what If I don't know what the tagging is yet and if it's even correct? To solve this, you would use the query op with a root ord filter. So, what this looks like is:

```
read(haystackConn).haystackCall("query",{root:"station:|slot:/Power",base:"neql",filter:"hs:point"})
```

Breaking down the axon call.

1. read(haystackConn) – This gives me a HaystackConnector to pass into the haystackCall function (https://skyfoundry.com/doc/ext-haystack/funcs#haystackCall)
2. The haystackCall then asks for the op name, in this case it's "query".
3. Lastly, it wants a grid of information which is the parameter list
    a. root – Niagara ord and/or slotPath to where to start the station, in this case it's station:|slot:/Power
    b. base – This tells the query what the query language is, in this case it is NEQL.
    c. filter – This is the formatted NEQL or BQL query to use.

The order of the arguments is not important to function properly.

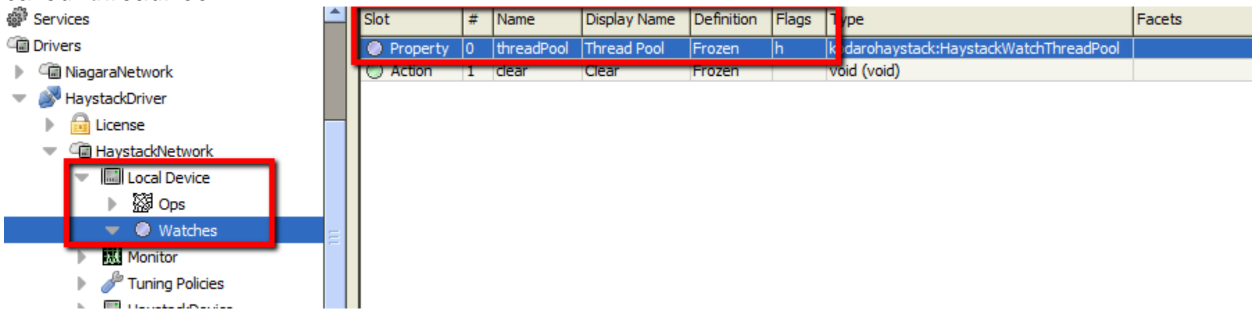
To better show the difference between this and read, this could have been written as:

```
read(haystackConn).haystackCall("read",{filter:"hs:point and hs:power"})
```

The difference here being it always assumes NEQL and there is no site level ord filter. This also assumes every point you want has been properly tagged with hs:power.

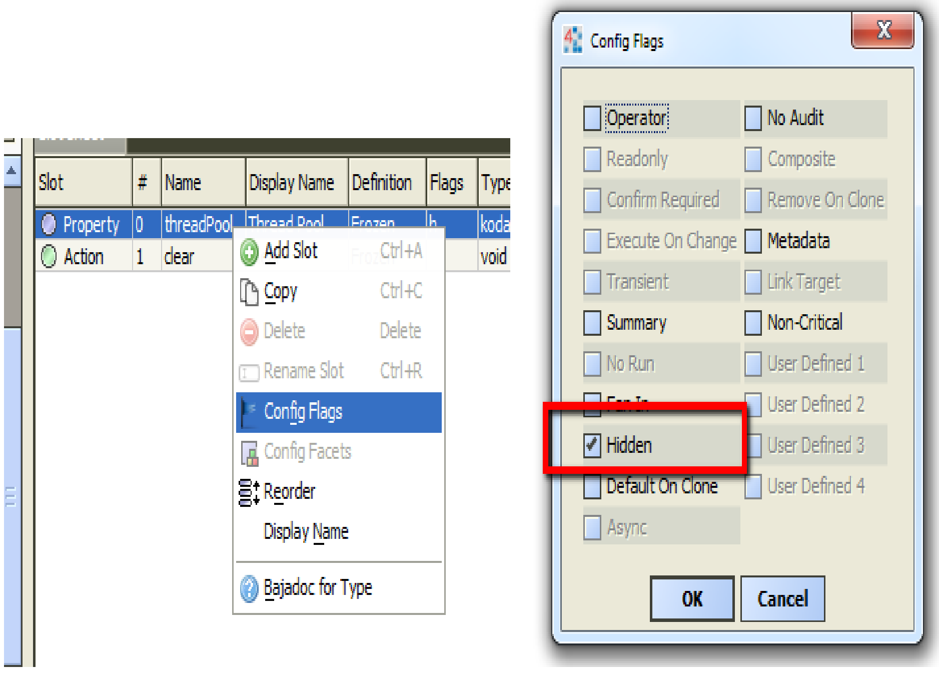
Watch Subscription Tuning

On a large system, there may be a reason you need to allocate more resources to the haystack watch section of this driver. This will only be the case if you are requiring more than 20 active watches simultaneously. By going to the slot sheet of the LocalDevice>Watches, you will see a hidden property called "threadPool".



Slot	#	Name	Display Name	Definition	Flags	Type	Facets
Property	0	threadPool	Thread Pool	Frozen	h	kodarohaystack:HaystackWatchThreadPool	
Action	1	clear	Clear	Frozen		void (void)	

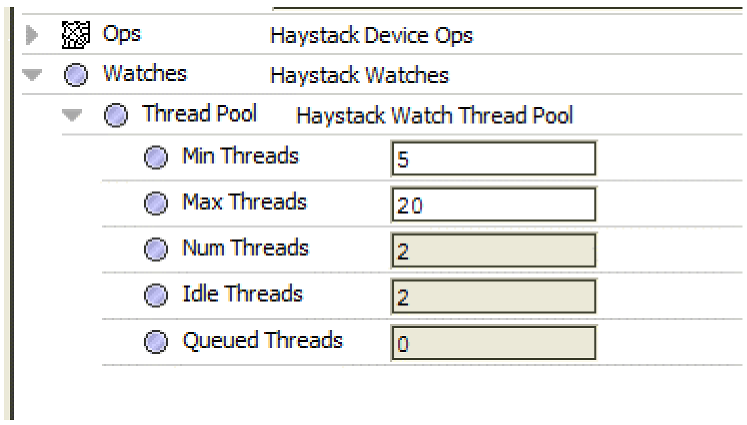
From there you want to right click and select "Config Flags" to bring up a dialogue to unhide this property by unchecking the "Hidden" flag.



Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	threadPool	Thread Pool	Frozen	h	koda
Action	1	clear				void

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

Once it has been unhidden, you can now view this on the property sheet of the watches object. Here you can monitor and/or configure the amount of active watches allowed.



Ops	Haystack Device Ops
Watches	Haystack Watches
Thread Pool	Haystack Watch Thread Pool
Min Threads	5
Max Threads	20
Num Threads	2
Idle Threads	2
Queued Threads	0

Every watch will spawn a new thread to subscribe to its points in the Niagara station. This means that the Max Threads property needs to be at least as large as the number of watches you intend to have active at the same time.

HISTORY

January 22, 2018: 1.0.17.5

- Misc. updates for N4.3+ security enhancements.

November 09, 2017: 1.0.17.4

- Updated to latest haystack protocol to include ver:”3.0” in responses instead of ver:”2.0”.

November 09, 2017: 1.0.17.3

- Licensing model changed to allow web ops on LocalDevice object to always operate even without a license key.

October 30, 2017: 1.0.17.2

- Fix Web Op “Ops”. Prior to this release was only returning the last Op in the list instead of a full grid of all the ops available.

October 2, 2017: 1.0.17.1

- Add Network Permissions to allow outgoing connections with security enforced with Niagara 4.3.
- Re-compiled to work with latest javax.servlet api 3.1.0.

October 2, 2017: 1.0.17

- Update legacy logging from AX to N4. Fixes security issue now enforced with Niagara 4.3 releases.

September 27, 2017: 1.0.16.1

- Clean up miscellaneous error messaging during station start concerning missing classes.

September 26, 2017: 1.0.16

- Fixed - Relationships not correctly being resolved when created outside of the Haystack dictionary. This also fixes instances where no ID would be generated if the Haystack dictionary was not installed.

September 06, 2017: 1.0.15

- Fix the O (oh) vs zero issue in license product code

August 22, 2017: 1.0.14

- Better error handling during Alarm Resolving

August 21, 2017: 1.0.13.1

- Added Unit Mapping tr-hr in Niagara To tonrefhr in Skyspark

August 21, 2017: 1.0.13

- Addition of the HaystackAlarmWatch object

July 06, 2017: 1.0.12

- Encode Niagara facets into a Haystack Dictionary for more haystack compliant encoding
- Relation names added during tag encoding process

June 06, 2017: 1.0.11

- Add cancel method to exports
- Fix concurrent update issue when running multiple exports simultaneously. Only an issue with Skyspark 3 due to ID pattern change

May 22, 2017: 1.0.10

- Merge encoding methods for exports and web ops

May 02, 2017: 1.0.9

- Initial Release

February 08, 2017: 1.0.0

- Initial Beta release