Project Haystack

Introduction and Overview of the NHaystack Software Module for Niagara-based Systems

NHaystack Overview

NHaystack is an open-source Niagara AX module that enables Niagara stations (JACE and WebSupervisor) to serve data in the Project Haystack format, via a RESTful protocol. Using NHaystack, external applications receive data that includes essential meta data (tags) to describe the meaning of the data.

NHaystack automatically generates standard tags for all the ControlPoints in your system. This feature allows for connecting immediately to the Niagara Station via Haystack once the NHaystack module has been installed, without requiring any further configuration. It makes discovering the points in your station as easy as issuing a simple query.

NHaystack also streamlines the process of adding user-specified Haystack tags to Niagara systems by providing a GUI tool that allows users to add the tags directly to Niagara components. Once tags have been defined, the data associated with the Niagara components, including the tags, are available over the Rest communications interface. This combination of the tagging tool and the Haystack protocol "engine" reduces the effort involved in connecting Niagara data to external software applications.

The development of NHaystack has been funded by J2 Innovations. This document includes information provided by: https://bitbucket.org/jasondbriggs/nhaystack

Benefits

- The ability to include meta data tags as part of Niagara data structures allows external applications to automatically interpret the meaning of data acquired from a Niagara system.

- The Haystack HTTP protocol is efficient and includes features coalesce requests to minimize network traffic and message size.

- The NHaystack module includes extensive features under the covers to unify real time and historical data structures in Niagara, which greatly simplifies access to data, and presentation in third party applications.

- The Haystack HTTP protocol tightly defines the relationship between client and server machines, allowing for third-parties to integrate easily with your equipment and data.

Key Features

- Provides drop-in support for the Haystack protocol on an AX system.
- Unifies the Component and History namespaces
- Allows for arbitrary queries of the station based on Haystack tags
- Makes it easy to create a Site-Equip-Point Hierarchy view of your system.

1. Getting started with NHaystack

To get started, install nhaystack.jar into an AX station. Then open the nhaystack palette in Workbench, and drag-and-drop the NHaystackService onto the "/Services" folder of your station.

This is all you need to do to get rolling. Your station is now automatically serving up all its ControlPoint objects and Histories as haystack point recs, via the Haystack REST Api. Many of the tags that are defined as being associated with points, like kind, unit, tz, his, cur, etc. are automatically generated for you.

2. How point recs are generated

In Niagara AX, ControlPoints and Histories exist in separate namespaces. There is one object in the station database which represents the current state of the point, (including its current value, actions to command it, and so forth), and a different object to represent its historical log of timestamp/value pairs.

However, in Haystack, a point rec models both of these concepts in one object. A point rec can have both a cur tag, which indicates the point has capability for subscription to its real-time current value, and it can have a his tag, which indicates that a point is historized with a history log of timestamp/value pairs.

NHaystack handles this mismatch by unifiying the AX namespaces. It automatically maps ControlPoints and Histories together so that only one Haystack rec is generated, with both a curtag and a his tag. If only one of the objects is present, then either the cur tag or this his tag is generated, but not both. Lets look at a couple of examples showing how this works.

2.1 A SIMPLE STATION

First lets take a simple example of a Niagara station that has two ControlPoints in it , with no History collection for those ControlPoints. The station will look similar to this:

```
station:|slot:/AHU1/NumericWritable
station:|slot:/AHU1/BooleanWritable
history:/my_station/AuditHistory
history:/my_station/LogHistory
```

This station will automatically have four point recs: two for the ControlPoints, and two for the Histories. The first two will have a cur tag, but no his tag, and the second two will have a histag but no cur tag.

```
{point,cur} station:|slot:/AHU1/NumericWritable
{point,cur} station:|slot:/AHU1/BooleanWritable
{point,his} history:/my_station/AuditHistory
{point,his} history:/my_station/LogHistory
```

Now lets add a NumericCov extension to the NumericWritable and enable it. Whenever we make a structural change to a station, by adding or removing a Component or History, we must invoke an action on the NHaystackService called rebuildCache. Invoking rebuildCache causes the NHaystack Server to traverse through the entire station and rebuild all of its internal data structures, so it can keep track of how everything in the station is interrelated.

After we invoke rebuildCache, the station will look like this:

```
{point,cur,his} station:|slot:/AHU1/NumericWritable
{point,cur}     station:|slot:/AHU1/BooleanWritable
{point,his}     history:/my_station/AuditHistory
{point,his}     history:/my_station/LogHistory
----            history:/my_station/NumericWritable
```

The station still has the same four point recs. However, a his tag has been added to the NumericWritable point. If a hisRead on that point is performed, the NHaystack Server will fetch the timestamp/value pairs for the associated History. The History for NumericWritable does not have its own rec created, because it has been unified with the NumericWritable.

By the way, the id tags that are generated for these recs consist of a single character, either "c" or "h" (which indicates whether the rec comes from the ComponentSpace or HistorySpace), followed by a ".", and then followed by a Uri-friendly Base64 encoding of either the slotPath or the historyId.
As a final example from this simple station, lets delete the NumericWritable and then invokerebuildCache. The station now looks like this:

```
{point,cur} station:|slot:/AHU1/BooleanWritable
{point,his} history:/my_station/AuditHistory
{point,his} history:/my_station/LogHistory
{point,his} history:/my_station/NumericWritable
```

We still have four point recs. However, the first rec is now gone. In its place we have a pointrec for the NumericWritable History. Note that this rec has a different id than the rec that has disappeared.

2.2 A MORE COMPLEX STATION

Now lets look at a more complex station -- one that is similar to what one often sees on an AX Supervisor. A very frequent case for supervisors is that they just import lots of Histories from Jaces.

```
----        station:|slot:/Drivers/NiagaraNetwork/jace1/Histories/Remote_NumericWritable
----        station:|slot:/Drivers/NiagaraNetwork/jace1/Histories/Remote_BooleanWritable
----        station:|slot:/Drivers/NiagaraNetwork/jace2/Histories/Remote_NumericWritable
----        station:|slot:/Drivers/NiagaraNetwork/jace2/Histories/Remote_BooleanWritable
{point,his} history:/supervisor/AuditHistory
{point,his} history:/supervisor/LogHistory
{point,his} history:/jace1/NumericWritable
{point,his} history:/jace1/BooleanWritable
{point,his} history:/jace2/NumericWritable
{point,his} history:/jace2/BooleanWritable
```

This station has six recs: two for the station's own audit and log Histories, and four for the imported Histories. Note that the four NiagaraHistoryImport objects do not have a pointassociated with them.
Now lets change the station by importing the ControlPoints for the first jace, and runningrebuildCache. Here is what the station looks like now:

```
{point,cur,his} station:|slot:/Drivers/NiagaraNetwork/jace1/points/NumericWritable
{point,cur,his} station:|slot:/Drivers/NiagaraNetwork/jace1/points/BooleanWritable
----         station:|slot:/Drivers/NiagaraNetwork/jace1/Histories/Remote_NumericWritable
----         station:|slot:/Drivers/NiagaraNetwork/jace1/Histories/Remote_BooleanWritable
----         station:|slot:/Drivers/NiagaraNetwork/jace2/Histories/Remote_NumericWritable
----         station:|slot:/Drivers/NiagaraNetwork/jace2/Histories/Remote_BooleanWritable
{point,his}     history:/supervisor/AuditHistory
{point,his}     history:/supervisor/LogHistory
----            history:/jace1/NumericWritable
----            history:/jace1/BooleanWritable
{point,his}     history:/jace2/NumericWritable
{point,his}     history:/jace2/BooleanWritable
```

You'll notice that the two Histories from jace1 no longer have their own auto-generated rec. Instead, the two imported ControlPoints know about those Histories, and will export them viahisRead. NHaystack creates this linkage by analyzing the relationships between imported ControlPoints found under a NiagaraPointDeviceExt, imported NiagaraHistoryImports found under a NiagaraHistoryDeviceExt, and the Histories found in the station's HistorySpace.

3. Tagging via the "haystack" slot

For many use cases of NHaystack, you will not need to do any explicit tagging on the station. However, sometimes you want to actually add tags to the recs that are generated. NHaystack supports this via the following convention: if there is a slot called "haystack" on a Component, and that slot is of type nhaystack:HDict, then the tags which are saved in that slot are exported in the rec along with all the auto-generated tags. There is a Workbench FieldEditor for HDict which allows you to edit these tags. In addition, this FieldEditor shows you all the auto-generated tags (though you cannot edit them).

You can add this slot via the SlotSheet View for the Component. However this is very tedious when you have to do it to many Components, so nhaystack provides a better way. There is a Workbench View on NHaystackService that has an area that you can drag-and-drop Components into. The area says "Drag Components here to add a 'haystack' slot". When you drop Components there, they will have a nhaystack:HDict "haystack" slot added automatically.

By the way, you may be wondering why nhaystack:HDict doesn't just appear in the palette. The reason for this is that nhaystack:HDict is a baja:Simple, and sadly Simples cannot be placed in a palette.

Whenever you alter a tag with the FieldEditor, you usually need to run rebuildCache. Its best to just get in the habit of running it any time you change a tag or alter the structure of a station.
Note that tagging of Histories is not currently supported. A future version of NHaystack will probably support this.

4. How site and equip tags work

In addition to tagging point recs, you may also want to include site and equip recs in your station, and link everything together via siteRef and equipRef pointers. NHaystack does support creating this site-equip-point hierarchy. Let's see how it works.

4.1 SITE TAGGING

Lets use our simple example station from section 2.1, except we'll add an EnumWritable to it. Note that this time we have also listed an entry for the parent "AHU1" folder in our simplified view of the station.

```
----        station:|slot:/AHU1
{point,cur} station:|slot:/AHU1/NumericWritable
{point,cur} station:|slot:/AHU1/BooleanWritable
{point,cur} station:|slot:/AHU1/EnumWritable
{point,his} history:/my_station/AuditHistory
{point,his} history:/my_station/LogHistory
```

If you look in the nhaystack palette, you'll see that there are Components for Site and Equip. Drag a Site over and drop it anywhere in the ComponentSpace, and then run rebuildCache. The station will now look something like this:

```
{site}      station:|slot:/Richmond
----        station:|slot:/AHU1
{point,cur} station:|slot:/AHU1/NumericWritable
{point,cur} station:|slot:/AHU1/BooleanWritable
{point,cur} station:|slot:/AHU1/EnumWritable
{point,his} history:/my_station/AuditHistory
{point,his} history:/my_station/LogHistory
```

Site and Equip already have a "haystack" slot, so you don't have to add one via the SlotSheet. Bring up the HDict FieldEditor for the Site, you'll see that there are quite a few tags you can fill out, like the various "geo" tags.

4.2 EQUIPMENT VERSUS CONTROLLERS

In Niagara, Controllers are modeled as an object of type driver:Device. Generally speaking, all ControlPoints exist underneath one of these Devices.

Howerver, in nhaystack there is no one-to-one relationship between an equip rec and a Device. Any Device can have its points belong to more than one equip, and any equip can havepoints from more than one Device.

This flexibility allows us to get away completely from the network-centric view of the world that one finds in an AX Station. You can create representations of your data that reflect the real-world equipment on your site, rather than the layout of your controller network. The following two sections, 4.3 and 4.4, explain in more detail how this works.

4.3 EXPLICTLY TAGGING POINTS WITH EQUIPREF

Lets continue with our example station from 4.1, and create an equip which references outsite. Then lets have a point reference the equip.

First, drag an Equip over from the palette and place it anywhere in the station. Underneath the Site is a good place for it. Name it "TrashCompactor".

Then edit the equip and assign its siteRef so that it references "Richmond".
Then run rebuildCache. Next, edit "/AHU1/BooleanWritable", and give it an equipRef that references "TrashCompactor". Run rebuildCache again. The station will now look like this:

```
{site}                                  station:|slot:/Richmond
{equip,siteRef="/Richmond"}             station:|slot:/Richmond/TrashCompactor
----                                    station:|slot:/AHU1
{point,cur}                             station:|slot:/AHU1/NumericWritable
{point,cur,equipRef="/TrashCompactor"} station:|slot:/AHU1/BooleanWritable
{point,cur}                             station:|slot:/AHU1/EnumWritable
{point,his}                             history:/my_station/AuditHistory
{point,his}                             history:/my_station/LogHistory
```

We have now successfully created a very simple site-equip-point hierarchy. Our nav tree looks like this:

```
Richmond
     TrashCompactor
         BooleanWritable
```

4.4 IMPLICITLY TAGGING POINTS WITH EQUIPREF

We could now continue and mark every single point under "AHU1" as belonging to someequip. However, that is a lot of drudgery, and its also prone to error. NHaystack provides us with a better way to do this.

Drag an Equip from the palette, drop it under the "AHU1" folder, and call it "my_equip". Edit theequip so that its its siteRef references "Richmond".

Next, add an optional "Str" tag called navNameFormat to the equip, and set its value to "%parent.displayName%". The reason for doing this will soon become apparent.
Then rebuild the cache. Now observe what has happened:

```
{site}                                  station:|slot:/Richmond
{equip,siteRef="/Richmond"}             station:|slot:/TrashCompactor
----                                    station:|slot:/AHU1
{equip,siteRef="/Richmond"}             station:|slot:/AHU1/my_equip
{point,cur,equipRef="/AHU1/my_equip"}   station:|slot:/AHU1/NumericWritable
{point,cur,equipRef="/TrashCompactor"} station:|slot:/AHU1/BooleanWritable
{point,cur,equipRef="/AHU1/my_equip"}   station:|slot:/AHU1/EnumWritable
{point,his}                             history:/my_station/AuditHistory
{point,his}                             history:/my_station/LogHistory

Site-Equip-Point Nav Tree:
    Richmond
         TrashCompactor
             BooleanWritable
         AHU1
             NumericWritable
             EnumWritable
```

All of the points underneath AHU1, except BooleanWritable, have an auto-generated equipReftag which references "/AHU1/equip". This happens because, during the cache rebuild process, NHaystack noticed that those 2 points had an ancestor which had an equip child. Since thosepoints were not explicitly annotated with an equipRef, NHaystack automatically linked them to that ancestors Equip object.

By implicitly tagging points this way, we can automatically generate large numbers of equipReftags without having to visit every point.

By the way, the reason that we added the navNameFormat to the equip was so that it would show up in our nav tree as "AHU1", rather than "my_equip". By default the navName of any object is just its AX displayName, but you can rig the nav tree so that alternate names are used. This is important because you must always ensure that all the children of a nav tree item have a uniquenavName.

Lets duplicate the "AHU1" folder (an then rebuildCache of course) to show how easy creating large nav trees is.

Before we do though, we'll fix our BooleanWritable so it has a better navNameFormat. Edit the tags for the BooleanWritable and set its navNameFormat to something like "%parent.displayName%_%displayName%". By performing this step, we ensure that all of the points underneath "TrashCompactor" in the nav tree will have a unique navName. Now our nav tree looks like this:

```
Site-Equip-Point Nav Tree:
     Richmond
         TrashCompactor
             AHU1_BooleanWritable
         AHU1
             NumericWritable
             EnumWritable
```

Now simply duplicate AHU1 and run rebuildCache. Our station will look like this:

```
{site}                                  station:|slot:/Richmond
{equip,siteRef="/Richmond"}             station:|slot:/TrashCompactor
----                                    station:|slot:/AHU1
{equip,siteRef="/Richmond"}             station:|slot:/AHU1/my_equip
{point,cur,equipRef="/AHU1/my_equip"}   station:|slot:/AHU1/NumericWritable
{point,cur,equipRef="/TrashCompactor"}  station:|slot:/AHU1/BooleanWritable
{point,cur,equipRef="/AHU1/my_equip"}   station:|slot:/AHU1/EnumWritable
----                                    station:|slot:/AHU2
{equip,siteRef="/Richmond"}             station:|slot:/AHU2/my_equip
{point,cur,equipRef="/AHU2/my_equip"}   station:|slot:/AHU2/NumericWritable
{point,cur,equipRef="/TrashCompactor"}  station:|slot:/AHU2/BooleanWritable
{point,cur,equipRef="/AHU2/my_equip"}   station:|slot:/AHU2/EnumWritable
{point,his}                             history:/my_station/AuditHistory
{point,his}                             history:/my_station/LogHistory

Site-Equip-Point Nav Tree:
     Richmond
         TrashCompactor
             AHU1_BooleanWritable
             AHU2_BooleanWritable
         AHU1
             NumericWritable
             EnumWritable
         AHU2
             NumericWritable
             EnumWritable
```

5. Keeping your data set consistent

5.1 FIXING BROKEN REFS

Occasionally when you are working on setting up the relationships between recs in a station, you will delete a Component that has other Components referencing it. For instance, in the example in 4.4, if we delete "TrashCompactor", then both of our BooleanWritables will have an invalid equipRef.
To fix this problem, there is an action on NHaystackService called removeBrokenRefs that you can invoke. This action deletes all of the refs in the station that do not reference a valid rec. Each time a broken ref is deleted, a message is also generated in the station log telling you which Component was fixed.

In a future version of NHaystack, there will be a view on NHaystackService that will make it easier for you to find the broken refs and fix them.

5.2 USING TIMEZONE ALIASES

Sometimes an AX TimeZone (a.k.a BTimeZone) do not map cleanly into a Haystack TimeZone (a.k.a HTimeZone). This happens when the BTimeZone uses an offset-style TimeZone ID, like "GMT-5:00", rather than a valid Olson ID, like "America/New_York". When this occurs, you will see errors in your log output that look like this:

```
ERROR [11:59:01 04-May-13 GMT-05:00][nhaystack] Cannot create tz tag: Unknown tz: GMT-
05:00
```

In cases like this, NHaystack simply omits the tz tag for the historized point in question.
However, NHaystack also allows you to provide a custom mapping that overcomes this problem, via the "timeZoneAliases" folder on your BHaystackService. Go to the nhaystack palette in Workbench, drag a "timeZoneAlias" onto the "timeZoneAliases" folder, and configure it so that the bogus AX TimeZone ID, like "GMT-5:00", is mapped onto a proper HTimezone, such as "America/New_York". Now when NHaystack is attempting to generate the tz tag, it will know how to proceed when it encounters non-Olson timezones.

In a future version of NHaystack, there will be a view on NHaystackService that will help you find which timezones are invalid in your dataset, but for now you must go through your log output, find the "Cannot create tz tag" error messages, and copy-and-paste the offending TimeZone ID into a timeZoneAlias that maps onto the 'real' HTimezone.

Note that if the machine that you are running Workbench from is mis-configured, it will provide a default HTimeZone of the form "Etc/GMT-5" in your timeZoneAlias. You should not use the HTimeZones from the "Etc" region unless you are really sure of what you are doing. Instead, always use the proper geographic region-and-timezone, like "America/New_York".

6. How IDs are generated

Each haystack rec must have an ID that uniquely identifies the object that the rec models.

By default, nhaystack generates IDs based on the slotPath of components (or the history ID for histories that are not associate with a point). E.g:

```
slot:/Foo/SineWave1 --> C.Foo.SineWave1
history:/nhaystack_simple/AuditHistory --> H.nhaystack_simple.AuditHistory
```

In the above example, the "C" prefix stands for the ComponentSpace, and the "H" stands for the HistorySpace.

Note that IDs are bi-directionally encoded -- you can always recreate the slotPath from an ID and vice-versa.

Special characters in the slotPath are handled by replacing the "$" character with a "~" character. The sole exception to this rule is that "$20", which represents a single " " character, is replaced with a dash, "-", to improve the readability of the IDS. E.g:

```
slot:/Foo/Sine$2fWave1 --> C.Foo.Sine~2fWave1
slot:/Foo/Sine$20Wave1 --> C.Foo.Sine-Wave1
```

If there is a site-equip-point hierarchy created in the station, then nhaystack will use that to generate the ID rather than the slotPath. E.g. if the SineWave in the first example was tagged up with an equip and a site, then its generated ID might look something like this:

```
slot:/Foo/SineWave1 --> S.Carytown.AHU1.SineWave1
```

In this case, there are actually two IDs that can be used to resolve the rec -- the slotPath version, and the Site-Equip-Point version.

Note that older versions of nhaystack used a Base64 encoding of the slot path, which looks something like this:

```
c.c2xvdDovRm9vL1NpbmVXYXZlMg~~
h.L25oYXlzdGFja19zaW1wbGUvQXVkaXRIaXN0b3J5
```

Nhaystack no longer generates IDs with this form, but it can resolve them.

For more information on Project Haystack and NHaystack visit: http://project-haystack.org/

Join the open-source community working to streamline data integration by becoming a