white paper

Tagging Niagara Components using Standard Dictionaries and Ontologies

by ***Eric Anderson, Software Engineer, Tridium with Introduction by Therese Sullivan, Marketing Director, Tridium***

Standardized tagging is a topic of great interest to any Niagara user that wants to save time, achieve better outcomes, and future-proof your work for data analytics, machine-learning (ML), and AI. On Tridium's part, we are building more support into Niagara Framework® to make it easier and more intuitive for Niagara users to begin systematically tagging all their station components See Figure 1 for a timeline of tagging-related features added to Niagara 4.

Niagara Framework users are a community of experts in the field of designing customized user interfaces for end-users of smart systems and smart buildings. Tagging is a core skill for accommodating the efficient resolving and rendering of graphical components in a customized UI, as well as for preparing a Niagara station for data analytics and ultimately ML and AI approaches. There

This whitepaper is about how Niagara users can configure tag dictionaries for easy reuse and faster overall deployment of a tagged project. It will cover how to imply tags and to deploy tags at-scale across projects. With the right tips and tactics, you can make light-work of the job of tagging your Niagara station components.

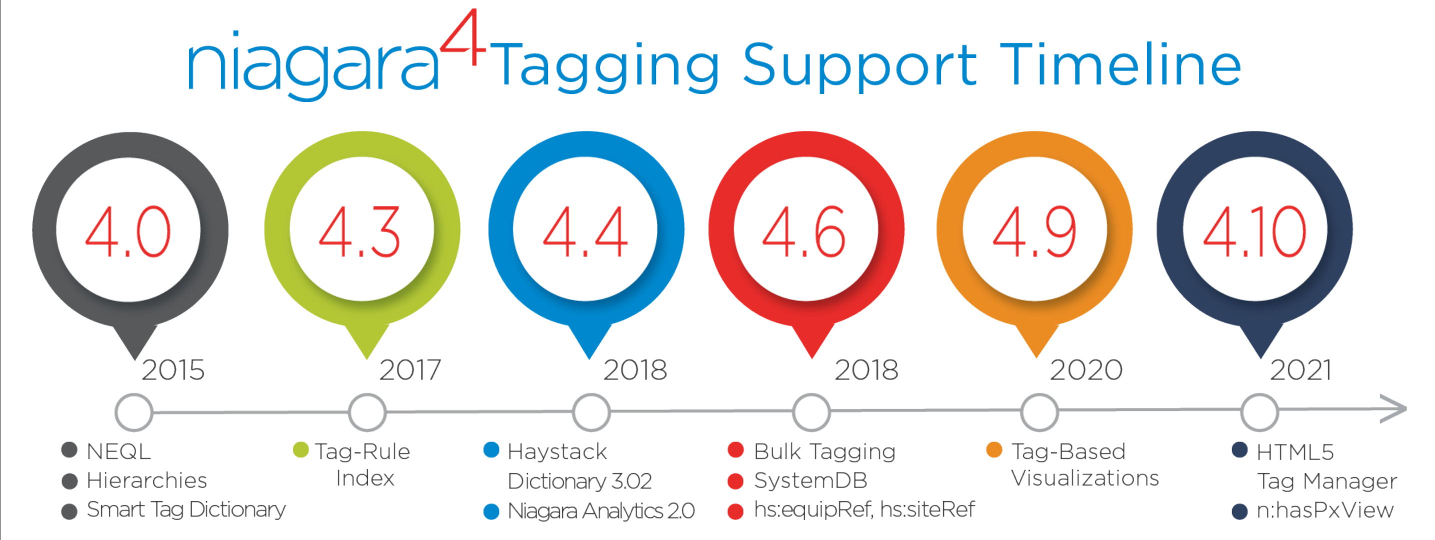


Figure 1. Introduced in Niagara4.0, Niagara entity query language (NEQL) made it possible to search for tags within Niagara stations . With every subsequent release, the Niagara development team has made tagging easier to comprehend, deploy and use.

is an industry-wide move toward standardized point tagging pioneered by open-source semantic-web organizations. These efforts began with Project Haystack, and now they include Brick Schema, Google Digital Buildings Project, and Buildings IOT Ontology Alignment Project. ASHRAE is also working on a standard ontology and approach naming. Architects, who are starting to think about controls strategies from the first design phases of new construction, are pushing for standardized tags that easily interoperate with BIM. Likewise, there are semantic web initiatives tied to International Energy Efficiency programs like Energy Star. All this activity is making property owners ask their systems integrators the question “How are you tagging our projects?” Today, the answer needs to be “We are using a standardized tagging methodology with the standard dictionaries you want.” Due to the tagging features built into the latest versions of Niagara Framework, Niagara users have the support they need to do just that. They can configure tag dictionaries for easy reuse and faster overall deployment of a tagged project.

This whitepaper will cover how to imply tags and to deploy tags at-scale across projects. There is special attention paid here to working with the Haystack dictionary and the NHaystack module — as it is a fact that Project Haystack was the first to start working on the semantic tagging challenge and Tridium has been part of the open-source working group supporting NHaystack. However, the tips and tactics described here have wider applicability. The authors’ goals - and the goals of Niagara Framework’s developers - are to help you make light-work of the job of tagging your Niagara station components — whatever the ontological approach and dictionaries brought to bear.

NHAYSTACK OVERVIEW

NHaystack is an open-source Niagara module that provides support for Project Haystack’s RESTful protocol in Niagara stations. The module was first available prior to the 2015 launch of Niagara 4, allowing points to be tagged even before tagging was supported natively in Niagara. The module also assisted with relating those points to site and equip components.

Niagara stations can act as Haystack servers by installing a NHaystackService. In addition to or instead of being a server, stations can connect to other Haystack servers by installing an NHaystackNetwork.

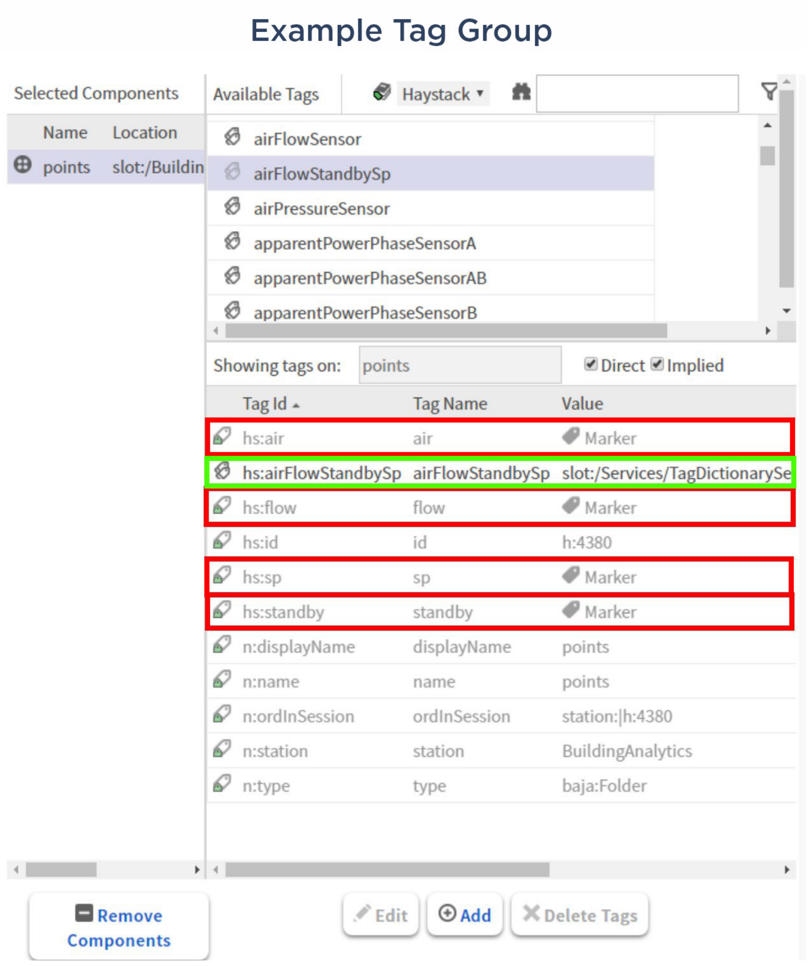
The latest version of the N4 NHaystack module supports Niagara Framework’s native tags and relations. The NHaystack GUI tool will reflect Niagara tags and relations on the component (for namespaces included in the NHaystackService’s Prioritized Namespaces property) and changes made in the GUI tool will update the component’s Niagara tags and relations (using the “hs” namespace). The N4 version of the NHaystack module is available on StackHub: [https://stackhub.org/package/nHaystack.](https://stackhub.org/package/nHaystack)

In the interest of cyber defense, Niagara now requires third-party modules to be signed by default. More specifically, Niagara Version 4.9 upped the module verification mode to “medium”, which requires modules to be signed by a valid, trusted certificate. The latest version of the NHaystack module (3.1.0.4.9) has been signed by Richard McElhinney using a certificate issued by one of the CAs in the system trust store. The module will be ready to use once dropped into the Niagara modules folder.

Custom ops available in the NHaystack module include ***Working-with-Niagara-Alarms*** and ***Working-with-Niagara-Schedules.*** The former returns alarms associated with a specified point, while the latter returns events in a specified weekly schedule.

The NHaystack project is always looking for contributions and feedback from the community. So please contact our working group if you have input. ▲

TRIDIUM



Name	Location
points	slot:/Buildin

Available Tags:
- airFlowSensor
- airFlowStandbySp
- airPressureSensor
- apparentPowerPhaseSensorA
- apparentPowerPhaseSensorAB
- apparentPowerPhaseSensorB

Showing tags on: points ☑ Direct ☑ Implied

Tag Id	Tag Name	Value
hs:air	air	Marker
hs:airFlowStandbySp	airFlowStandbySp	slot:/Services/TagDictionarySe
hs:flow	flow	Marker
hs:id	id	h:4380
hs:sp	sp	Marker
hs:standby	standby	Marker
n:displayName	displayName	points
n:name	name	points
n:ordInSession	ordInSession	station:|h:4380
n:station	station	BuildingAnalytics
n:type	type	baja:Folder

Figure 2: Highlighted in red in this HTML5 Tag Manager view are the tag group's tags. These are implied on the component that has a tag group relation to this tag group. Highlighted in green is an implied tag group id marker tag. Such marker tags are useful when searching a station for components on which this tag group is applied, instead of all components that contain the tag group's tags. A word of caution: if you delete a tag group definition or its dictionary, the relation will also be deleted.

DIRECT VS IMPLIED TAGS

All Tag Dictionaries in Niagara contain tag, tag group, and relation definitions. The tag and tag group definitions feed Niagara's HTML5 Tag Manager and Workbench Edit Tags dialog for applying direct tags and tag groups to components. Direct tag groups are an n:tagGroup relation from a component to a tag group definition. Figure 2 illustrates the result of adding a direct airflow standby set-point tag group to a component. Niagara's Haystack Tag Dictionary includes tag group definitions for each equip point grouping defined by Haystack.

Smart Tag Dictionaries can also include tag rules for implying tags, tag groups, and relations. Every direct tag, tag group, or relation uses a component slot, so using implied tags will save memory. Having tag rules centralized in the Smart Tag Dictionaries can make your tagging effort easier to maintain and update. Once you have a working set of tag rules, you can drop them into other stations, and tags will start to be implied immediately.

The trade-off with tag rules is that they do need to be computed at run time. To minimize this effort, Niagara 4 only evaluates the tag rules necessary for the tags being searched—not all tag rules in all dictionaries. However, you should understand this trade-off. Direct tags are faster for searching, but implied tags via tag rules are the way to go from the standpoint of storage efficiency and long-term maintenance.

TAG RULE CONDITIONS

Tag rules are made up of tag rule conditions; here are some simple ones:
- '**And**' condition is true if all child conditions are true. It is short-circuiting and will stop evaluating subsequent conditions, once the first condition returns false.
- '**Or**' condition is true if any child conditions are true. It is short-circuiting and will stop evaluating once the first condition returns true.
- '**Always**' condition is useful when you want to imply items to every component in the station, regardless of that component's type or any other tags on that component.

• ‘**IsType**’ condition checks to see that the component is the specified type or one of its subclasses, before applying the tag.

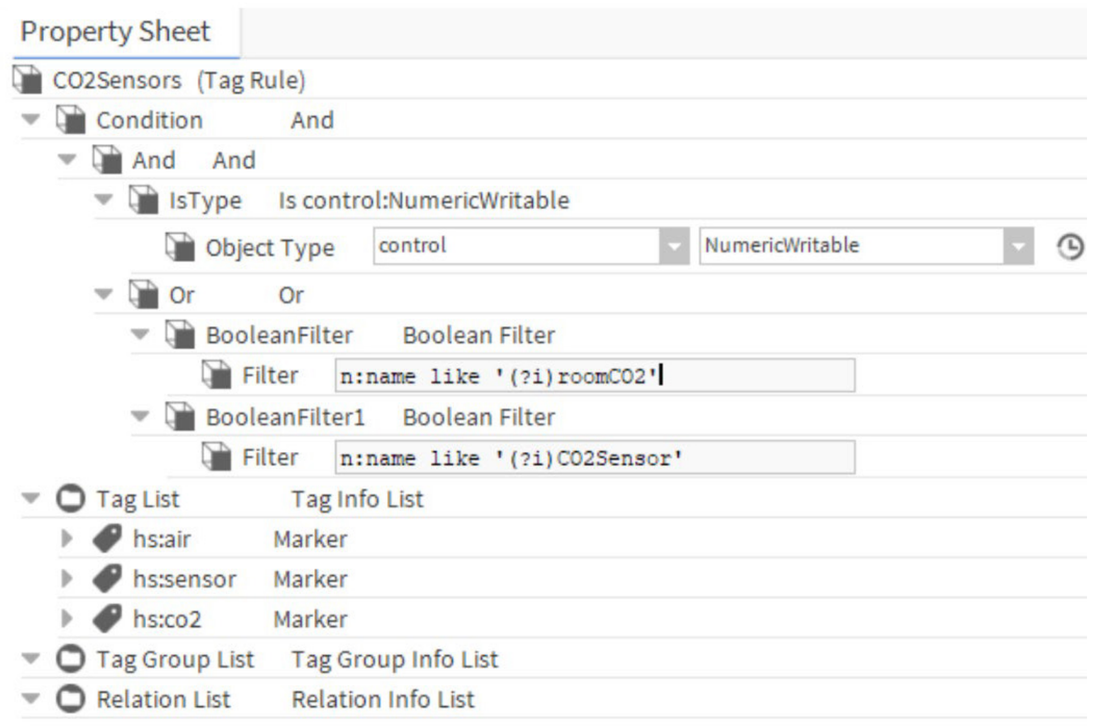
The Niagara4 tag dictionary palette offers another class of conditions around ‘**BooleanFilter**.’ The basic ‘**BooleanFilter**’ applies an NEQL query to each component. If the query applies to the component, then that condition returns true. The ‘**BooleanFilter**’ has two subclasses:

• ‘**HasAncestor**’ evaluates the NEQL query on the component itself, and then on its ancestors. As soon as it finds one ancestor that satisfies the query, then that condition returns true.

• ‘**HasRelation**’ evaluates the query on the component itself and then on any component it can reach using a specified relation id.

‘**IsType**’ conditions can be evaluated quickly, thus it is best to put those conditions near the top of a tag rule. ‘**IsType**’ will match the specified condition and subclasses. If you need an exact type match, you can use a ‘**BooleanFilter**’ to apply an NEQL query that uses ‘n:type’ tags. For example, the query might be written: “**n:type = ‘control:numericPoint’**”, if you want just numeric points and not numeric writables. Use the ‘**like**’ operator to compare a string value to a regex expression. You can make that regex case-insensitive by including ‘**(?i)**’ at the beginning. For example, you could form a query like “**n:name like ‘(?i)roomCO2’**”. Figure 3 illustrates a Tag Rule example that uses a mix of these condition types in combination.

Tag Rule Example


Property Sheet

CO2Sensors (Tag Rule)	
Condition	And
And	And
IsType	Is control:NumericWritable
Object Type	control / NumericWritable
Or	Or
BooleanFilter	Boolean Filter
Filter	n:name like '(?i)roomCO2'
BooleanFilter1	Boolean Filter
Filter	n:name like '(?i)CO2Sensor'
Tag List	Tag Info List
hs:air	Marker
hs:sensor	Marker
hs:co2	Marker
Tag Group List	Tag Group Info List
Relation List	Relation Info List

Figure 3. This tag rule has at the top an ‘And’ condition with two child conditions underneath it. The first is the ‘IsType’ so it’s looking for numeric writables. If it finds a numeric writable then it will evaluate the ‘Or’ condition. The ‘Or’ has two Boolean Filters that search for and compare ‘n:name’ tags. If either of those filters apply to a component, then the implied tags in that tag list – air, sensor, CO2 – will be implied on the component.

SIMPLE VS SMART IMPLIED TAGS

There are two flavors of implied tags, and Niagara users doing tagging can benefit from both. Simple implied tags have an implied value that is fixed. Examples include marker tags and value tags where the value is specified. Consider the Haystack tag '**hs:phase**': if this tag is in a rule and it is set to 'BC,' then when that tag is implied, it will always have the string value 'BC.' Likewise, '**hs:stage**' = 1.00 will always have the value of one.

The other flavor of implied tags are smart implied tags. The following tags derive their value from the component they're implied on:

- '**n:name**': the value of the tag is derived from the name of the component.
- '**n:type**': the value of the tag is derived from the type spec of the component
- '**hs:kind**': from the Haystack Tag Dictionary, the value of this tag is derived from the type of point that it is implied on.
- '**hs:tz**': from the Haystack Tag Dictionary, the value is derived from the time zone of the station

The following tags may or may not be implied based on the component:

- '**n:input**': tag will be implied only on non-writable points
- '**n:output**': tag will be implied only on writable points.
- '**n:hasPxView**': tag will only be implied if the component has a PX view associated with it

The following tags may or may not be implied based on the component and their value is derived from the component:

- '**n:history**': tag will be implied if a component has an enabled history extension. The value of the tag will be the history ID defined in the extension.
- '**hs:enum**': from the haystack dictionary, the value of this tag will be implied on Boolean and Enum points. For Boolean points, the value of the tag will be "false, true" or will use the "falseText" and/or "trueText" facet values should those be present. For Enum points, the tag value will be set using the range facet value, if that is present.

NIAGARA HAYSTACK TAG DICTIONARY

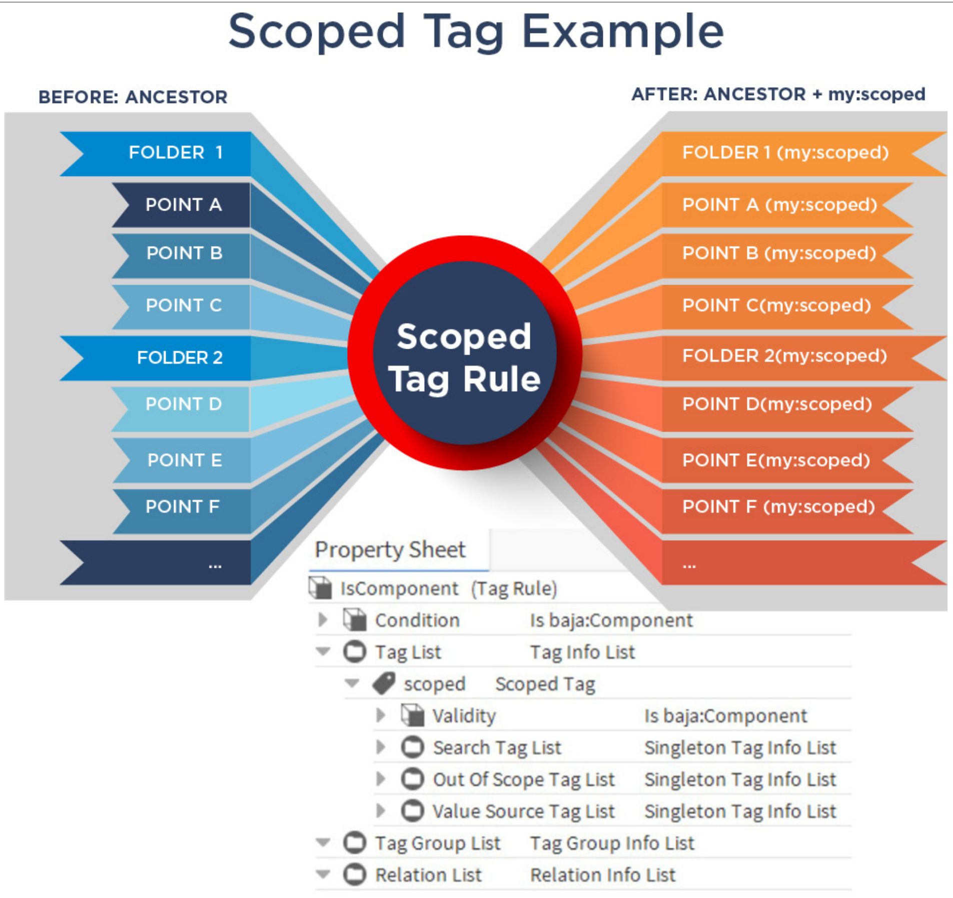
Tridium's Niagara Haystack Tag Dictionary provides general support for Haystack tags and relations in a Niagara station. It is contained within the haystack-rt module and includes all of the tags defined in Project Haystack version 3.0.2. A tag group is defined for each set of equip points. For example, the "dischargeAirTempSensor" tag group can be set on the appropriate VAV point and the "discharge", "air", "temp", and "sensor" tags will be implied on that component. A relation is defined for each ref tag.

In practice, to place an "ahuRef" tag on a component representing a VAV, for example, in Niagara, you would create a relation from that component to the component that represents the AHU with which it is associated. When relations like this are exported by the NHaystack module, they are converted to ref tags with their value set to the "id" tag value of the relation endpoint. Finally, there are tag rules defined to imply tags and relations in common situations.

Here are some examples of implied marker tags in the Haystack Tag Dictionary:

- "**connectio**n" on components of type "driver:DeviceNetwork"
- "**cur**" and "**point**" on components of type "**control:ControlPoint**"
- "**writable**" on components of type "**control:IWritablePoint**"
- "**device"** on components of type "**driver:Device**"
- "**hvac**" on components that have certain direct or implied tags, such as "ahu", "vav", "chiller", etc.

Something to watch out for when working with implied values is stale **curVal** tags. Implying a "**curVal**" tag does not cause a point subscription, so the tag's value can become out-of-date. To query the station for more up-to-date values, a point must be placed in subscription by, for example, adding a point extension. In the Niagara System Database, the tag's value will be the one captured at System Index time. System Indexing usually occurs once a day (indexing more frequently to keep **cur** values updated is not recommended.) Instead of querying for **cur** values, a **WatchSub Op** can be used. ▲



IsComponent (Tag Rule)	
Condition	Is baja:Component
Tag List	Tag Info List
scoped	Scoped Tag
Validity	Is baja:Component
Search Tag List	Singleton Tag Info List
Out Of Scope Tag List	Singleton Tag Info List
Value Source Tag List	Singleton Tag Info List
Tag Group List	Tag Group Info List
Relation List	Relation Info List

Figure 4. In this Scoped Tag Rule Example, the ID of the Scoped Tag is 'scoped.' It is in a dictionary with the namespace 'my.' The effect of applying the 'my:scoped' tag to a component named 'Ancestor' is to place the 'my:scoped' tag on all of its descendants. The implied tags are marker tags because the 'my:scoped' tag applied to this ancestor component is also a marker tag.

- '**hs:maxVal**' and '**hs:minVal**' functions are similar to hs:enum in that they will resolve to the component's maximum and minimum facet values
- '**hs:unit**' is based on the unit facet set for the component.

SCOPED TAGS

The '**scoped**' tag is another type of smart implied tag. The simplest use case for a '**scoped**' tag is to imply a tag based on whether a given component has an ancestor with a tag that has the same ID as the scoped tag. The value of that implied tag will match the ancestor tag's value. Figure 4 shows a tag rule built using a scoped tag.

The Niagara 4 tag dictionary palette has some advanced options for scoped tags as well: You can search for a different tag ID rather than matching the ID of the 'scoped' tag. You can copy the value of a tag other than the one being searched for. Finally, you can specify an '**out-of-scope**' tag ID. If a tag with that ID is found before finding the tag being searched for, then the scoped tag will not be implied (Figure 5).

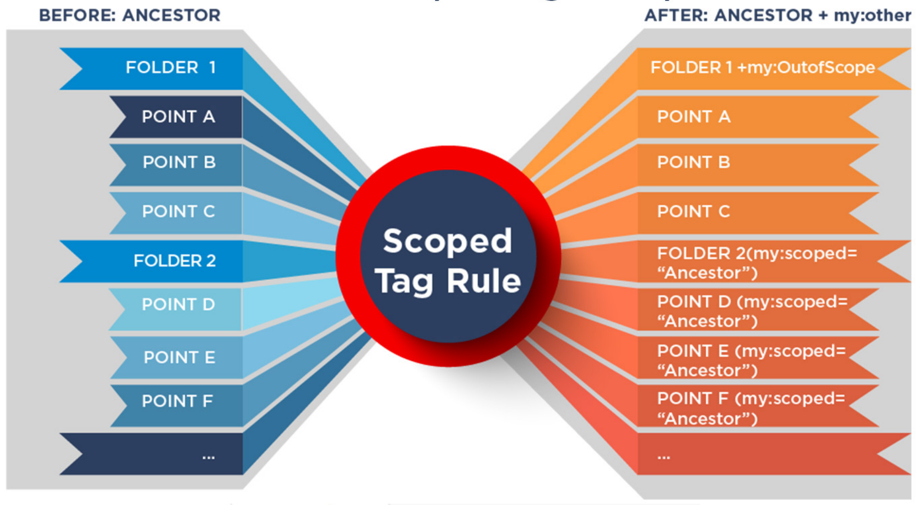
SCOPED TAGS FOR SYSTEM INDEXING

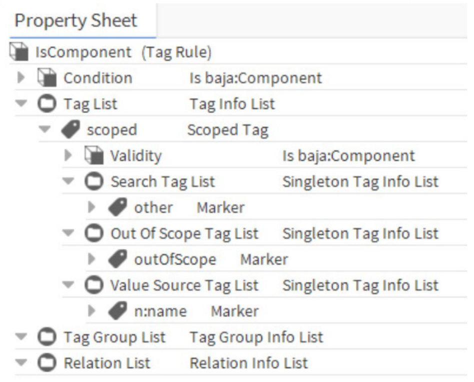
One use case for scoped tags in a Niagara station is to support system indexing. Under the tag dictionary palette, there is a system index dictionary that you can drag into your tag dictionary service. It contains two marker tags: '**excluded**' and '**included**'. The tag dictionary also includes a tag rule that implies an 'excluded' tag. By default, components with the '**systemIndex:excluded**' tag are not indexed.

IMPLIED RELATIONS

In addition to smart implied tags, the Niagara dictionary palette also offers smart implied relations. The '**n:child**' and '**n:parent**' smart relations help to navigate a component tree. If you're on a driver network, for example, you can use the '**n:childDevice**' relation to get all the child devices under that network. If you are at one of those devices, you can navigate to the network by traversing the '**n:parentNetwork**' relation. On a device, you can use the '**n:childPoint**' and '**n:childNullProxyPoint**' relations to get from the device to all the child points. The '**n:parentDevice**' relation can get you back from a child point to the parent device it belongs to.

Advanced Scoped Tag Example

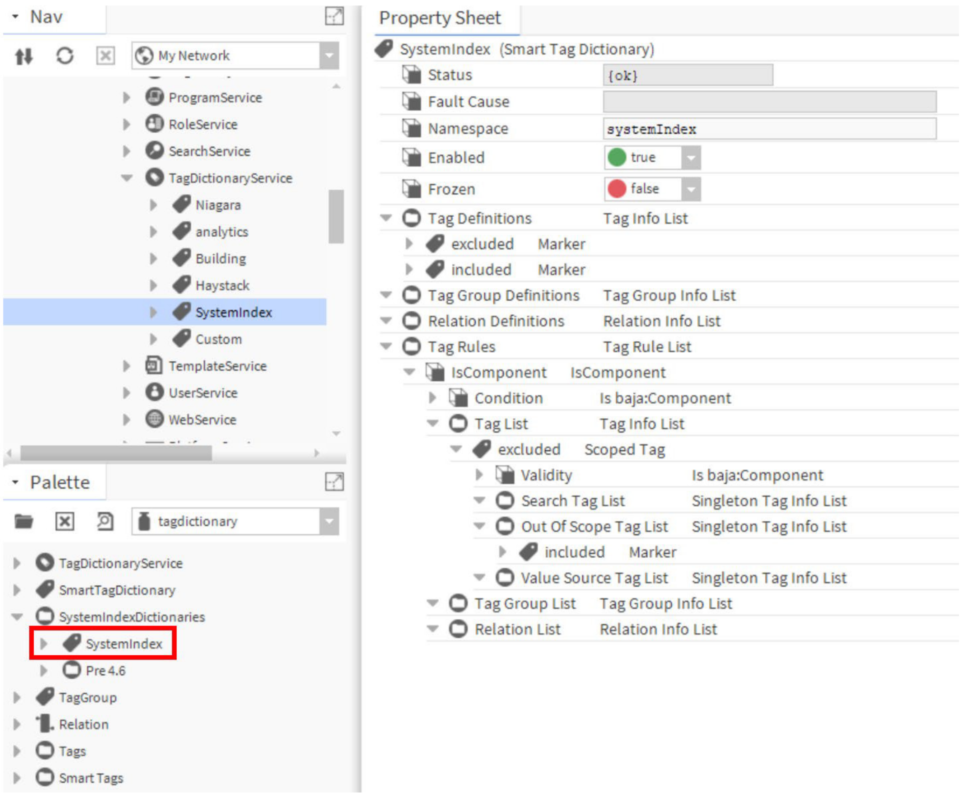

BEFORE: ANCESTOR
FOLDER 1
POINT A
POINT B
POINT C
FOLDER 2
POINT D
POINT E
POINT F
...
Scoped Tag Rule
AFTER: ANCESTOR + my:other
FOLDER 1 +my:OutofScope
POINT A
POINT B
POINT C
FOLDER 2(my:scoped= “Ancestor”)
POINT D (my:scoped= “Ancestor”)
POINT E (my:scoped= “Ancestor”)
POINT F (my:scoped= “Ancestor”)
...



```
Property Sheet
IsComponent  (Tag Rule)
  Condition                Is baja:Component
  Tag List                 Tag Info List
    scoped                 Scoped Tag
      Validity                     Is baja:Component
      Search Tag List              Singleton Tag Info List
        other      Marker
      Out Of Scope Tag List        Singleton Tag Info List
        outOfScope     Marker
      Value Source Tag List        Singleton Tag Info List
        n:name     Marker
  Tag Group List           Tag Group Info List
  Relation List            Relation Info List
```

Figure 5. In this advanced Scoped Tag Rule example, instead of searching for a tag with the ID ‘my:scoped’, the rule searches for a tag with the ID ‘my:other.’ An out-of-scoped ID is specified: ‘my:outOfScope’.. Instead of copying the value of the tag being searched for, the value of an ‘n:name’ tag will be copied. When a ’my:other’ tag is applied to the ancestor component, Folder 2 and its descendants have the ‘my:scoped’ tag implied on them; the value of those implied tags is copied from the value of the ‘n:name’ tag, which in this case is the string ‘Ancestor’. Because Folder 1 has the ’my:outOfScope’ tag on it, the ‘my:scoped’ tag is not implied on it or any of its descendants.

Example System Index Dictionary Service



```
Nav
My Network
  ProgramService
  RoleService
  SearchService
  TagDictionaryService
    Niagara
    analytics
    Building
    Haystack
    SystemIndex
    Custom
  TemplateService
  UserService
  WebService

Palette
tagdictionary
  TagDictionaryService
  SmartTagDictionary
  SystemIndexDictionaries
    SystemIndex
    Pre 4.6
  TagGroup
  Relation
  Tags
  Smart Tags
```

```
Property Sheet
SystemIndex  (Smart Tag Dictionary)
  Status                   {ok}
  Fault Cause
  Namespace                systemIndex
  Enabled                  true
  Frozen                   false
  Tag Definitions          Tag Info List
    excluded     Marker
    included     Marker
  Tag Group Definitions    Tag Group Info List
  Relation Definitions     Relation Info List
  Tag Rules                Tag Rule List
    IsComponent     IsComponent
      Condition            Is baja:Component
      Tag List             Tag Info List
        excluded     Scoped Tag
          Validity                 Is baja:Component
          Search Tag List          Singleton Tag Info List
          Out Of Scope Tag List    Singleton Tag Info List
            included     Marker
          Value Source Tag List    Singleton Tag Info List
      Tag Group List       Tag Group Info List
      Relation List        Relation Info List
```

Figure 5. When preparing for a system index, you can apply the ‘excluded’ marker tag to the top of that part of the tree you want to exclude, and the ‘excluded’ tag will be implied on all descendants. The system index will ignore those components. If you want to run a system index that includes a subtree of the tree that you are excluding, you can put the ‘included’ marker tag on the top of that section.

Niagara 4's Haystack Dictionary offers '**hs:equipRef**' and '**hs:siteRef**' relations. These smart relations will be implied between any non-null proxy points and an ancestor that has the '**hs:equipRef**' tag on it. If that equip component also has an '**hs:siteRef**' relation to a component with the '**hs:site**' tag, the '**hs:siteRef**' relation will be implied from those non-null proxy points to that equip's site. These smart implied relations are time savers when building a station by avoiding the tedious addition of many direct relations.

TAG-BASED PX GRAPHICS & SYSTEMDB

Niagara tag-based PX graphics use all the tag and relation information that one can build into a station as described above. Introduced in Niagara 4.9, tag-based PX graphics use bindings based on NEQL queries. Bound components must have the expected tags and relations; however, they don't need to have the same names, nor do they need to be located at the same place in the component tree. Niagara will find them just based on tags and relations. NEQL traverse queries can be used to create bindings to anywhere in the station using relations. The great advantage of tag-based PX graphics is that they are easier to reuse across stations.

Another feature in Niagara that uses tag and relation information is System Database.

Niagara System Database (systemDB) is our name for the stored result of System Indexing – the periodic update of the tag and relation information for selected entities from each station in your Niagara network. By default, Niagara indexes all networks, devices, points, schedules, point/device folders. A System Index will also pull up any components with a PX view using the new '**n:hasPxView**' implied tag. SystemDB is currently single-tier, and it enhances any features that use NEQL queries. You can search all the stations in your Niagara network and build hierarchies against the results, given that they have been indexed up to systemDB. As of 4.10, if your Niagara Supervisor has a systemDB and you've indexed your stations to it, you can leverage virtual tag-based PX graphics against that system database.

SUMMARY

Since the Niagara Framework is tag-agnostic, users have the utmost flexibility when applying tags to Niagara stations and building systems. The result is visualizations that are reusable and manufacturer- and equipment-agnostic. Tag-based bindings streamline on-site efficiency, save time and provide more flexibility when deploying graphic templates on new and existing Niagara stations. ▲

About the Author:
Eric Anderson, a Software Engineer, has been working on tagging and tag hierarchies since he joined Tridium in 2015. Tridium created and continues to enhance the Niagara Framework®, an open platform that facilitates system integration and control.

TRIDIUM
tridium.com

Americas:	support@tridium.com
EMEA:	ordersEMEA@tridium.com
Asia-Pacific:	tsupportAP@tridium.com

2021 - 0020