

Technical Document

Niagara Tagging Guide

December 19, 2024

niagara⁴

Legal Notice

Tridium, Incorporated

3951 Western Parkway, Suite 350
Richmond, Virginia 23233
U.S.A.

Confidentiality

The information contained in this document is confidential information of Tridium, Inc., a Delaware corporation (Tridium). Such information and the software described herein, is furnished under a license agreement and may be used only in accordance with that agreement.

The information contained in this document is provided solely for use by Tridium employees, licensees, and system owners; and, except as permitted under the below copyright notice, is not to be released to, or reproduced for, anyone else.

While every effort has been made to assure the accuracy of this document, Tridium is not responsible for damages of any kind, including without limitation consequential damages, arising from the application of the information contained herein. Information and specifications published here are current as of the date of this publication and are subject to change without notice. The latest product specifications can be found by contacting our corporate headquarters, Richmond, Virginia.

Trademark notice

BACnet and ASHRAE are registered trademarks of American Society of Heating, Refrigerating and Air-Conditioning Engineers. Microsoft, Excel, Internet Explorer, Windows, Windows Vista, Windows Server, and SQL Server are registered trademarks of Microsoft Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Mozilla and Firefox are trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, and Sedona Framework are registered trademarks, and Workbench are trademarks of Tridium Inc. All other product names and services mentioned in this publication that are known to be trademarks, registered trademarks, or service marks are the property of their respective owners.

Copyright and patent notice

This document may be copied by parties who are authorized to distribute Tridium products in connection with distribution of those products, subject to the contracts that authorize such distribution. It may not otherwise, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Tridium, Inc.

Copyright © 2025 Tridium, Inc. All rights reserved.

The product(s) described herein may be covered by one or more U.S. or foreign patents of Tridium.

For an important patent notice, please visit: <http://www.honpat.com>.

Contents

- About this Guide**7
 - Document change log7
 - Related documents9
- Chapter 1. Tagging Overview**11
 - License requirements11
 - Tagging process11
- Chapter 2. Common tagging tasks**13
 - Creating a tagged device13
 - Adding Ad Hoc tags14
 - Removing a tag14
 - Add tags to objects in the Discovered pane15
 - Add tags in the Database pane16
 - Adding a Tag Group to a component16
 - Creating a custom tag group17
 - Adding a tag to an existing tag group18
 - Adding tags using Batch Editor19
 - Editing tags in a template20
 - View implied tags using Edit Tags dialog21
 - Viewing implied tags using Spy view22
 - Selecting or exiting tag mode (manager views)23
 - Exporting and importing tag dictionaries24
 - Creating a new tag dictionary24
 - Editing a tag dictionary exported to CSV25
 - Importing a tag dictionary in CSV format28
 - Exporting a tag dictionary30
- Chapter 3. Tag dictionary service**33
 - Smart tag dictionary33
 - Haystack smart tag dictionary33
 - Modifying the Haystack tag dictionary35
 - Creating the Haystack tagsImportFile and equipImportFile36
 - Editing tags in the Haystack tagsImportFile and equipImportFile37
 - Configuring the Haystack dictionary to auto-import modifications39
 - haystack-HsTagDictionary39
 - Haystack Tags Import File format40
 - Haystack Equip Import File format42
 - haystack-EquipRelation43
 - haystack-SiteRelation43
 - H4TagDictionary43
 - Migrating to Haystack 444
 - Migrating Haystack 3 items44
 - Haystack 4 import47

Tags	48
Choice tags	48
Tag group	49
Relations	51
Tag rules	53
Updating an existing H4 tag dictionary	55
Brick tag dictionary	57
Updating Brick tag dictionary	59
Brick custom rules	63
Chapter 4. Tagging reference	67
About tags	67
Online tagging versus offline tagging	68
About the Edit Tags dialog	68
Relation Manager	70
Components and views in the tagdictionary module	72
tagdictionary-TagDictionaryService	73
Neqlize options	76
Tag Rule Index	77
Implied tags index	80
NiagaraTagDictionary	82
About tag dictionaries	82
tagdictionary-SmartTagDictionary	83
tagdictionary-SystemIndex	84
Tag Definitions (TagInfoList)	85
Tag Group Definitions (TagGroupInfoList)	87
Relation Definition (RelationInfo)	88
Tags (SimpleTagInfo)	89
Smart Tags	90
tagdictionary-NameTag	90
tagdictionary-DisplayNameTag	91
tagdictionary-TypeTag	91
tagdictionary-historyIdTag	91
tagdictionary-HistoryMarkerTag	91
tagdictionary-ScopedTag	92
SystemDb usage example	93
Hierarchy QueryLevelDef usage example	95
tagdictionary-SingletonTagInfoList	97
tagdictionary-TagRuleList	97
Tag Rule components	98
Conditions	102
tagdictionary-Always	102
tagdictionary-And	102
tagdictionary-BooleanFilter	102
tagdictionary-HasAncestor	103
tagdictionary-HasRelation	103
tagdictionary-IsType	103

tagdictionary-Or	104
tagdictionary-OrdScope	104
tagdictionary-DataPolicy	105
Tag Dictionary Manager view	107
HTML5 Tag Manager view	108
Tag Manager view	109
Relation Manager view	111
Chapter 5. Glossary	113

About this Guide

This topic contains important information about the purpose, content, context, and intended audience for this document.

Product Documentation

This document is part of the Niagara technical documentation library. Released versions of Niagara software include a complete collection of technical information that is provided in both online help and PDF format. The information in this document is written primarily for Systems Integrators. To make the most of the information in this book, readers should have some training or previous experience with Niagara software, as well as experience working with JACE network controllers.

Document Content

This guide explains to the Systems Integrator how to use the Tagging feature.

Document change log

Updates (changes and additions) to this document are listed below.

December 19, 2024

- Added "Updating an existing H4 tag dictionary" to "Haystack 4 import" chapter.
- Added "Relation Manager" to "Tagging reference" (as of Niagara 4.15)
- Added "Relation Manager view" to "Components and Views" (as of Niagara 4.15)
- Updated "Brick tag dictionary" topic (as of Niagara 4.15)

April 10, 2024

- Added new "NotRule" to "Tag Rule components" topic (Niagara 4.14).

October 5, 2023

- Added new "Brick tag dictionary" and "Updating Brick tag dictionary" topics (as of Niagara 4.14).
- Added "tagdictionary-SmartTagDictionary" component (as of Niagara 4.14).

October 4, 2022

- Added new "H4TagDictionary" topic to the "Haystack smart tag dictionary" chapter.

July 19, 2022

- Added new topic "Tag Manager" to the "Tagging Reference" chapter.

April 26, 2022

- In the "Tag Dictionary Manager" view, added that the list of tag dictionaries displays version information and the **New** button limits the creation of a new tag dictionary to a "smart tag dictionary".

November 2, 2020

- Added information on the HTML5 Tag Manager view.
- To support online help, updated ID values on "Rules" and "Tag Rule Component" topics and combined the child component details in the "Tag Rule Component" topic.

February 4, 2020

- Edited the "TagDictionaryService" component topic to add information on several new properties related to tag-based NEQL Ords functionality (in Niagara 4.9 and later).

August 6, 2019

- Edited Haystack component topics to add information on changes in Haystack tag dictionary import

functionality (in Niagara 4.4U3, Niagara 4.7U1, Niagara 4.8 and later), and changes in the components provided in the palette.

May 23, 2019

- Edited the component topics, "tagdictionary-HasAncestor" and "tagdictionary-HasRelation", to provide additional details.
- In the Tagging Reference chapter, added separate component topics for "haystack-EquipRelation" and "haystack-SiteRelation" to support online help.

November 30, 2018

- Added several procedures on working with Haystack tagdictionary import files.
- Edited the component topics, "tagdictionary-TagDictionaryService" and "tagdictionary-ScopedTag", added information on SingletonTagInfoList frozen properties and usage examples of same to the "tagdictionary-ScopedTags" topic.
- Added the component topic, "haystack-HsTagDictionary" and reference topics, "Haystack Tags Import File format" and "Haystack Equip Import File format".
- Edited the "Tag Dictionary Manager" view topic.

Updated: March 26, 2018

Updated for functional changes in Niagara 4.6:

- Added note to the "tagdictionary-TagDictionaryService" topic describing two new smart tag dictionaries which can be used to exclude portions of the station from the system indexing process.
- Added component topics, "tagdictionary-SystemIndex" and "tagdictionary-scopedTag".

Updated: October 24, 2017

Updated for functional changes in Niagara 4.4:

- Many changes throughout, all to do with the removal of the TagDictionary component and the frozen slot on the TagDictionaryService named Monitor (and related functionality) from the tagdictionary module.
- Removed the following component topics from this guide: "tagdictionary-TagDictionary" and "tagdictionary-TagGroupMonitor".

Updated: August 30, 2017

- In the topics, "Tag Rule Index" and "Implied Tags Index," there are added notes clarifying the type of memory used, and type of tags that may be indexed."
- This update includes many structural changes throughout the "Tagging Reference" section.
- In the "Tagging components" section of this guide, added several new component topics.

Updated: January 12, 2017

Updated for Niagara 4.3:

- Edited the topic, "About the Tag Dictionary Service", added content describing the new Tag Rule Index feature .
- Edited the topic, "Tag Rules", added a section describing the new functionality in tag rules, includes new topics on the "Tag Rule Index", "Implied Tags Index", and "Scoped Tag Rule" features.
- Added glossary entries on these features.

Updated: November 3, 2016

- Edited the topic, "Creating a custom tag group", to provide information on best practices in tag group naming.
- Edited "Tag Definitions" topic to provide Tag component property descriptions.
- Edited the "Tag Group Definitions" topic to add section on tag group handling when editing tags from

within various views.

Updated: July 26, 2016

Added the procedure, "Creating a custom tag group".

Updated: July 14, 2016

Added information on changes for Niagara 4.2, which include the following:

- Incorporated minor changes throughout to support branding.
- In the "Tag Rules" topic, added new information on TagRules in the tagdictionary palette.

November 29, 2015

Added information on changes supporting the Tag Dictionary Service functionality for Niagara 4.1. The following topics have been modified as described.

- Creating a tagged device: edited content in steps 4 and 5. Results info explains that tags in added tag groups are replaced with an implied relation.
- "Editing tags in a template", added note to step 5.
- "About the Edit Tags dialog", deleted duplicate content in 2nd paragraph and added note at end that describes changed handling of tags in tag groups.
- "Tag Definitions", added note that about data policy in tag definitions.
- "Tag Group Definitions", added content that describes changed handling of tags in tag groups, data policies, and other instances.
- The following topics have been added to this guide: "Adding a Tag Group to a component", "Adding a tag to an existing Tag Group", "Data Policies", "Tag Group Monitor".

Initial publication: August 31, 2015

Related documents

Following documents provide information related to using tags.

- Niagara Hierarchies Guide
- Niagara Relations Guide
- Niagara Templates Guide

Chapter 1. Tagging Overview

Adding tags to your data model can streamline the process of setting up a system, especially large or enterprise systems. Instead of manually mapping data into the application point by point, trend by trend, systems integrators can use tags to facilitate the process. Tag information can also facilitate and improve search results and hierarchical navigation design. Tagging is a form of semantic modeling that assigns information (one or more tags) to objects. The tag information can help integrators and users significantly when searching for objects, designing system structures or navigating hierarchies.

If you add tags to station objects using standard Tag Dictionaries, other applications can discover station content without having to understand the naming convention used by the installer or system integrator. Typical station tagging might include things such as: networks, devices, points, control blocks, and more. You can also map all of these example objects to domain-specific semantic entities such as, buildings, systems, equipment to further indicate how they relate to each other. Tagging can identify a device and indicate where it is physically located. By identifying and locating devices, tags provide a context for the device that can be used in many different ways. When you use tags, you can reduce or eliminate the requirement to manually map objects directly to a desired application.

License requirements

The **tags** license is required to use the `TagDictionaryService` and tag dictionaries on a station. The `Dictionary.limit` attribute limits the number of tag dictionaries available for the system. Any dictionaries added above the limit for the license will be in fault. When a dictionary is in fault, the tags in that dictionary are not available in the **Edit Tags** dialog. By default, you are limited to the first two tag dictionaries. However, the `Dictionary.limit` attribute is configurable on the license in the same manner as are device limits.

For more licensing information, see licensing topics in the Niagara Platform Guide.

Tagging process

This section describes the basic tagging process. Before adding tags, make sure that you have the dictionaries that you need to complete the process.

The basic process for tagging involves the following:

1. Identify your purpose. Possible uses could be one or more of the following examples:
 - Enterprise structure navigation: In this case you may want to focus on using tags that include geographical information.
 - Systems maintenance views: In this case you may need to use tags that include device or equipment information.
 - End user navigation: In this case you may use functionally related tags.

2. Make sure you have the dictionaries you need.

In many cases, the Niagara Tag Dictionary may be sufficient. By default, it is in the `TagDictionaryService` folder in a new station. You can add the **Haystack dictionary** from the **haystack** palette if needed. You can also create Ad Hoc tags or create your own custom dictionary if you wish to.

NOTE: For new stations, you may need only the Niagara and Haystack Smart Tag Dictionaries. However, you can reduce or eliminate your tagging efforts by looking for Smart Tag Dictionaries developed by the Niagara community.

3. Add tags to your components.

You can add tags one at a time or you can use Tag Groups (containers of various tags) to add multiple tags with each **Add** action. You can add tags during or after a discovery process and you can also use the Batch Editor to add tags.

NOTE:

Adding a tag group adds a relation between the component and the tag group definition. The tags in the group are implied on the component. If you change the tags in the group, the revised set of tags are implied on the component.

Chapter 2. Common tagging tasks

The following sections include descriptions of some common ways to use tagging.

Creating a tagged device

You can add Direct Tags to a device (or other station objects) to provide additional semantic information. You may add more than one type of tag to a device to support multiple hierarchical navigation schemes. You can also use Tag Groups to add a predefined collection of tags to the device in a single add action.

Prerequisites:

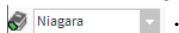
- One or more installed tag dictionaries. If necessary, add required tag dictionaries to the `TagDictionaryService`.

NOTE: If tagging offline, it is possible that no dictionaries are available. In that situation the system searches for tag dictionaries in alternate locations.

This task describes how to use the **Edit Tags** dialog box to add individual tags or tag groups from an installed tag dictionary.

Step 1. Right-click on the device that you want to tag and select **Edit Tags** from the popup menu.

Step 2. In the **Edit Tags** dialog box, select a dictionary from the option list in the top left corner



TIP: In the **Search** field, you can use a shortcut to designate the dictionary. Type `hs:` for Haystack, `n:` for Niagara, and similarly for other dictionaries.

The top half of the dialog box shows a list of tags available from the selected dictionary.

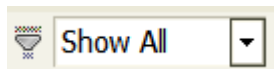
Step 3. Use the filter fields as needed to limit the number of tags displayed. For example:

- Type in the **Search** field



to filter by tag name. Tags are filtered immediately as you type.

- Select an option from the option list



to filter based on validity options (`Show All`, `Valid Only`, or `Best Only`).

Step 4. Add any number of tags to suit your needs (for example, `n:device`, `hs:geoState`, `my:bldgRef`) using either of the following methods:

- To add an individual tag from a tag dictionary, select one or more tags in the **Tag Dictionary** (upper) pane and click **Add Tag** to assign the selected tag(s) to the device.
- To add a predefined collection of tags from a tag dictionary, in the **Tag Dictionary** (upper) pane in the dialog, scroll down to **Tag Groups** and select a tag group, and click **Add Tag** to assign the selected collection of tags at once.

NOTE:

Adding a tag group adds a relation between the component and the tag group definition. The tags in the group are implied on the component. If you change the tags in the group, the revised set of tags are implied on the component.

The assigned individual tags and added tag groups are listed on the **Direct Tags** tab in the lower half of the dialog.

- Step 5. Edit any tag value fields, as appropriate, and click the **Save** button to save the added tag assignments.
- Step 6. Optional: For tags that have Ord type values such as `hs:siteRef`, refer to the following steps as an example of how to add a link to your tag.
 - a. Click the option list arrow located to the right of the tag value field.
 - b. Select the appropriate link type from the options menu.
 - c. Browse to the desired link and select it.
 - d. Select the `Handle` option and click **OK**.

Result

The device is now tagged.

Adding Ad Hoc tags

You can add Ad Hoc Tags to any station object to provide additional semantic information without using an installed tag dictionary. Ad Hoc tags are tags that you create directly from the **Edit Tags** dialog box. These tags are not found in any tag dictionary.

Ad Hoc tags are useful for development or testing purposes, allowing you to test without adding or modifying tag dictionaries and without using the tags that are already in use by active production applications. However, when applying tags that are used by applications, best practice is to use tags from standardized tag dictionaries that are applied system-wide.

- Step 1. Right-click the component that you want to tag and select **Edit Tags** from the popup menu.
- Step 2. In the **Edit Tags** dialog box, and without making any selections click the **AddTag** button. The **Add Tag** dialog box appears.
- Step 3. In the **TagId** field, enter a new tag name using the following syntax: namespace:tagname. For example, `my:datalogs`. For best practices, use a consistent naming convention. Also, it is important to use a namespace that does not conflict with that of other installed tag dictionaries.
- Step 4. In the **Type** field, use the option list to select the tag type from the options available.

NOTE: In your Ad Hoc tag, do not use a namespace that is identical to an existing tag dictionary. For example, do not use `hs:`, `n:`, or other namespace characters that would conflict with existing tag dictionaries.

For an Ad Hoc tag with the TagId `my:datalogs` you could select a type called `baja:String` to determine that the tag value be a String type of data.
- Step 5. To assign the tag to your selected component, click **OK**. The new tag is added and appears in the **Direct Tags** table in lower half of the dialog box.
- Step 6. Edit any tag value fields (for example, String, Ord,), as appropriate, and click **Save** to save the tag assignments.

Result

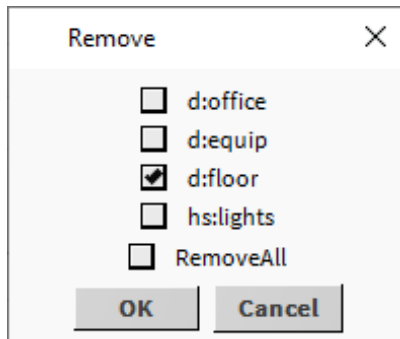
The component is now tagged.

Removing a tag

You can remove direct tags from individual station objects using the **Remove** dialog box.

NOTE: This task does not apply to implied tags.

- Step 1. Right-click the object that you want to edit and choose **Edit Tags** from the menu.
- Step 2. From the **Edit Tags** dialog box, click **Remove Tag**.
The **Remove** dialog box appears showing a list of all the Direct Tags that are assigned to the selected component.



- Step 3. Select the individual tags that you want to remove or choose **Remove All** and then click **OK**.
The selected tags are removed from the table listing under the **Direct Tags** tab in the lower half of the dialog.
NOTE: Check that the appropriate tags are now listed in the **Edit Tags** lower pane, under the **Direct Tags** tab. Your deletions are not complete until you click **Save**. If you want to revoke the delete action, click **Cancel**.
- Step 4. To complete the task, click **Save**.

Result

The tag is removed from the selected object.

Add tags to objects in the Discovered pane

Tagging is integrated into the **Station Manager** and **Point Manager** views to make it easier to tag editable stations or points in the **Discovered** pane before adding them to the **Database** pane.

Prerequisites:

Device Manager or **Point Manager** view is active with Tag Mode selected. Points or devices are discovered and listed in the **Discovered** pane.

Tagging during discovery is optional but it is a convenient way to add metadata as you add points or devices. This task describes how to add tags only. It does not describe all point or device fields that need to be reviewed or edited during an add process.

- Step 1. From the **Point Manager** or **Station Manager** view, in the **Tag Dictionary** pane, select the desired tag dictionary.
- Step 2. Select one or more discovered objects in the **Discovered** pane.
- Step 3. In the **Tag Dictionary** pane, select one or more tags to add and click **Add**.
- Step 4. In the **Add** dialog box, check that the appropriate tags are added (table in top pane) and add values to any tags that have editable fields.
- Step 5. Click **OK** and verify that your tags appear with the desired objects in the **Database** pane.

Result

Devices or points are added to the station with tags.

Add tags in the Database pane

Tagging is integrated into the **Station Manager** and **Point Manager** views to make it easier to tag editable stations or points that are in the **Database** pane.

Prerequisites:

- **Device Manager** or **Point Manager** view is active with **Tag Mode** selected.
- Points or devices are listed in the **Database** pane.

Tagging objects that are in the Database pane of a manager view is a convenient way to add metadata to your points or devices. This task only describes how to add tags, it does not describe point or device fields that may be edited from the manager view.

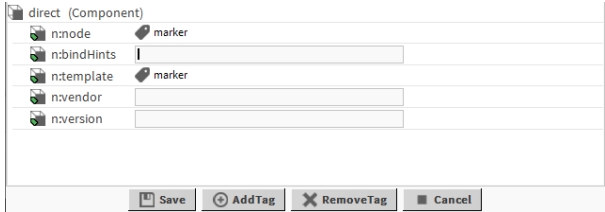
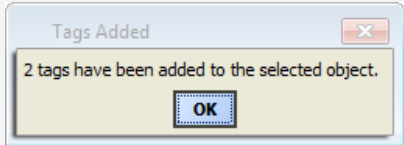
Step 1. From the **Point** or **Station Manager** view, in the **Tag Dictionary** pane, select the desired tag dictionary.

Step 2. Select one or more objects in the **Database** pane.

Step 3. In the **Tag Dictionary** Pane, select one or more tags to add and click the **TagIt** button. Depending on the type of tag you are adding, one of the following happens:

- If one or more tags have a value field, an **Tags Edit** dialog box opens and displays all fields.
- If no tags have value fields, a **Tags Added** dialog box displays a confirmation message indicating how many tags are added.

Step 4. Depending on the type of tags you have added, do one of the following:

Option	Description
Edit tag values and click OK in the Tags Edit dialog box.	 <p>If this dialog box displays, then you have tag values to edit. Edit the value fields and click OK.</p>
Click OK in the Tags Added dialog box.	 <p>If this dialog box displays, no tag values are available. Click OK.</p>

Adding a Tag Group to a component

Adding a Tag Group allows you add a predefined collection of tags to a component in a single action. Typically, tags are in a tag group because it is common for each of the tags to be assigned to the same component. The **Device Manager** and **Point Manager** views of a driver, and the **Edit Tags** dialog are the primary methods for adding a tag group to a component. This procedure describes how to use the **Edit Tags** dialog to add a tag group.

Prerequisites:

- One or more installed tag dictionaries. If necessary, add required tag dictionaries to the **TagDictionaryService**.

NOTE: If tagging offline, it is possible that no dictionaries are available. In that situation the system searches for tag dictionaries in alternate locations.

Step 1. Right-click on the component that you want to tag and select **Edit Tags** from the popup menu.

Step 2. In the **Edit Tags** dialog box, select a dictionary from the option list in the top left corner



The top half of the dialog box shows a list of individual tags available from the selected dictionary.

Step 3. Scroll down to see the list tag groups in the dictionary.

TIP: To limit the number of tags displayed, use the **Search** and the **Filter** fields as needed.

Step 4. To assign the selected collection of tags, select the desired tag group and click **Add Tag**. The assigned tag group displays as an Ord on the **Direct Tags** tab (lower half).

Step 5. To save the added tag assignments. click **Save**.

Result

Once you save the added tag assignments, the set of tags in the tag group are implied tags on the component.

Creating a custom tag group

Use the tagdictionary palette to create a custom tag group in a tag dictionary. This procedure describes how to edit an existing tag dictionary, add a custom tag group and configure it with tags.

Prerequisites:

- Existing tag dictionary
- The tagdictionary palette is open.

Once a tag group is applied to an object, it implies all of the individual tags in its tag list, as well as implying a marker tag that bears the name of the tag group. This allows you to easily define a NEQL search for the marker tag for that tag group rather than define a search by concatenating each of the tags in the tag group's tag list.

Note that a recommended best practice for naming tag groups is to use a name that reflects its intended usage. For example, the Haystack outsideAirTempSensor tag group is typically applied to points representing outside air temperature.

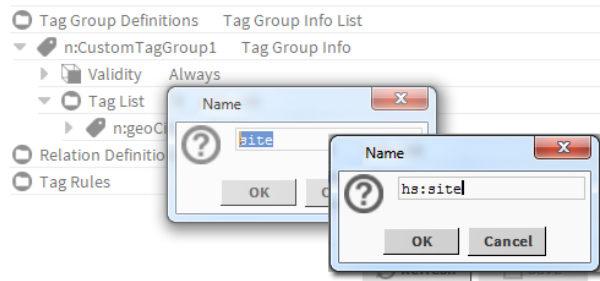
Additionally, for tags that you add to a tag group a best practice is to use only "fully qualified" tag names. This means, add tags from a tag dictionary's existing tag definitions and include the namespace of the source tag dictionary in the added tag's name. This ensures that the tags will resolve correctly for NEQL queries.

Step 1. Expand the **TagDictionaryService** node, double-click the tag dictionary you intend to edit to open the **Property Sheet** view.

Step 2. From the palette, drag a **TagGroup** component onto the dictionary's **Tag Group Definitions** property and in the resulting popup, enter a name for this tag group (see recommended best practices above) and click **OK**.

Step 3. Expand your new tag group and proceed with the following steps:

- a. Drag any previously defined tag one at a time from any tag dictionary's tag definitions to the **TagList** sub-property in the new tag group.
- b. In the **Name** popup, edit the tag name to prepend its source dictionary's namespace (ex.: change site to **hs:site**, as shown).



- c. Click OK.
- d. Repeat the steps a–c until you are finished adding tags to this tag group.

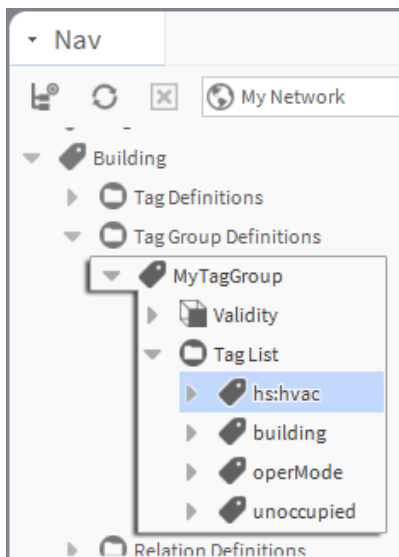
Result

On completion, the new custom tag group is immediately available for use. Changes made in this procedure are automatically saved to the tag dictionary (selected in step 1).

When applying the new tag group to an object, the multiple tags included in the group are implied at once on the object, as well as a marker tag bearing the name of that tag group.

Adding a tag to an existing tag group

You can add a tag from a different tag dictionary to an existing tag group. Optionally, you can add a tag from the tagdictionary palette. Shown in the following image, the `hs:hvac` tag (copied from the Haystack tagdictionary) is added to `MyTagGroup` in the Building tagdictionary.



Prerequisites:

- At least two tag dictionaries are installed.
- One of tag dictionaries contains a TagGroup.

Step 1. In the Nav tree, expand the tag dictionary to the TagGroup that you wish to edit.

Step 2. Expand the second tag dictionary to select a Marker tag that you wish to add to the TagGroup in

the first tag dictionary.

- Step 3. Drag (or right-click and copy) the selected tag, and drop it (or right-click and paste it) on the TagList folder of the Tag Group Definition that you edit.

NOTE: An alternative is to add a Marker tag from the tagdictionary palette.

- Step 4. In the **Name** window, enter the tag's desired name as a fully qualified tag name including the namespace with the colon separator. For example: `hs:hvac`.

NOTE: There is no verification that the tag name entered is actually defined in a tag dictionary. If it is not defined in a tag dictionary, the added tag is an "ad hoc" tag. While you can certainly use ad hoc tags, the recommended tagging best practice is to use tags that are contained in a standardized tag dictionary that is applied system-wide.

Changes to the tag group are saved automatically.

Result

This namespace overrides the application of the parent dictionary's defined namespace. The added tag automatically becomes an `n:tagGroup` relation (an implied tag) from the component to the corresponding tagdictionary's TagGroupInfo.

Adding tags using Batch Editor

You can add Direct Tags to large numbers of objects using the Program Service, **Batch Editor** view. Use the **Batch Editor** to locate objects that need tagging and use the **Add Tags** button in the **Batch Editor** view to add tags.

Prerequisites:

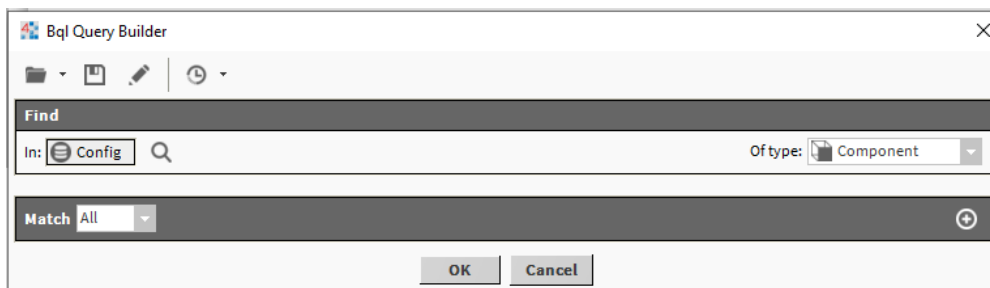
- One or more installed tag dictionaries. If necessary, add required tag dictionaries to the TagDictionaryService.

NOTE: If tagging offline, it is possible that no dictionaries are available. In that situation the system searches for tag dictionaries in alternate locations.

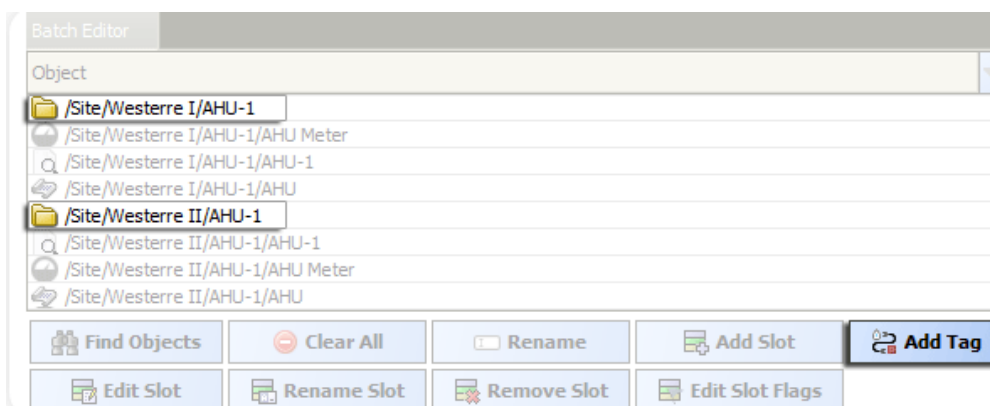
This task describes how to use the **Edit Tags** dialog box to add individual tags or tag groups from a dictionary that is installed in the station Services folder.

NOTE: If you are using tags to support multiple hierarchical navigation schemes, you may add more than one type of tag to a component. You can use tag Ggroups for adding multiple tags in a single add action.

- Step 1. In the station Nav tree, expand the **Station > Config > Services** and double-click **Program Services**.
- Step 2. In the **Batch Editor** view, click **Find Objects** and use the **Bql Query Builder** to produce a list of objects that you want to tag.



The search produces a list of matching components.

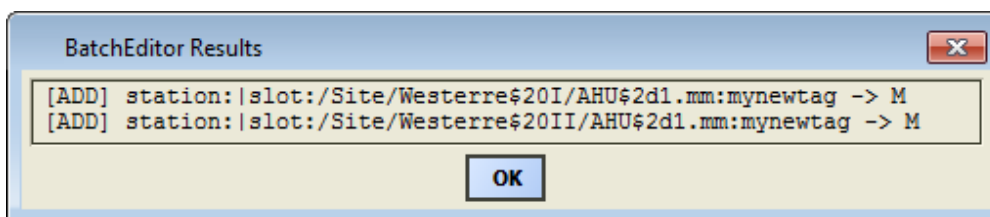


Step 3. Select and remove the unwanted components in the table and click **Add Tag**.

Step 4. Select one or more tags from the **Tag Dictionary** (top) pane (or you may select **Add Tag** from this dialog window to apply custom tags) and click **AddTag** to assign the selected tags to all components.

NOTE: Select a Tag Group, if appropriate, to add several tags at once.

The selected tags are added and appear in the **Direct Tags** table in the lower half of the dialog box. Finally, the **BatchEditor Results** dialog opens with all tag actions listed.



Result

The components are tagged.

Editing tags in a template

You can edit an existing template to add additional direct tags to the objects in it, or to remove or modify the existing tags. These changes are made on the template **Configuration** tab

Prerequisites:

- An existing template
- One or more installed tag dictionaries

This task describes using the **Template** sidebar to access an existing template and invoke the **Edit Tags** window from the template **Configuration** tab.

Step 1. To locate a template, open the **Template** side bar by clicking **Window > Side Bars > Template**.

Step 2. In the **Template** side bar, click the pull down menu and select either: the **Template** or **Module**.

NOTE: To see templates stored in a template module, select the `myModule` folder and expand the module. You cannot edit a template stored in a module. When you open it, `ReadOnly` appears in the top left corner of the **Template View**. To make changes you must first click **Save As** and save it as a new template in the `templates` folder.

Step 3. Double-click on the template.

The **Template View** opens that displays the template tabs with the **Template Info** tab selected.

Step 4. Click the **Configuration** tab in the **Template** view, right-click the object you want to change in the left pane, and select **Edit Tags**.

The **Edit Tags** dialog displays.

Step 5. Proceed to add tags, remove tags, or modify values of existing tags and click **Save** to close the **Edit Tags** dialog.

Step 6. To save your changes to the template when finished, click **Save** or, click **Save As** to create a new variation of the template with a different filename (leaving the original template unchanged).

Next steps

NOTE: For more details on using the **Template** view **Configuration** tab, refer to the [Template Guide](#) sections "Creating a template" and "Template reference".

View implied tags using Edit Tags dialog

Viewing implied tags can be useful when designing a custom tag dictionary to confirm that certain objects are getting the desired tags, or when designing a NEQL query for a hierarchy definition or search. Implied tags do not appear in an object Property Sheet or other typical views. One way to view these tags is on the **Implied Tags** tab in the **Edit Dialog** box.

Prerequisites:

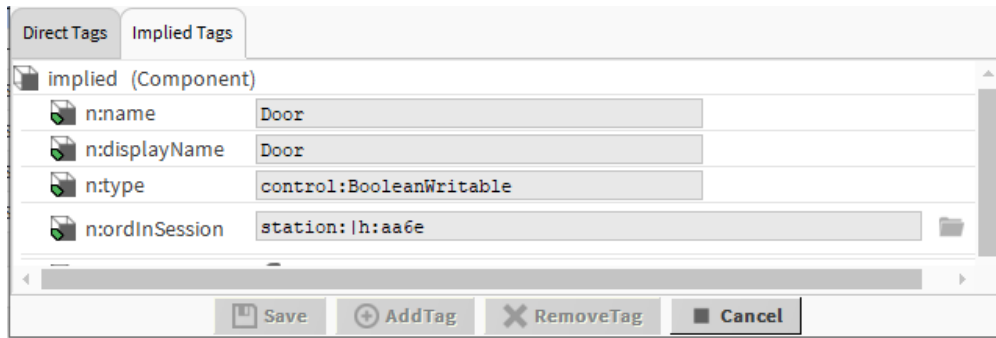
- One or more installed tag dictionaries. If necessary, add required tag dictionaries to the `TagDictionaryService`.

NOTE: If tagging offline, it is possible that no dictionaries are available. In that situation the system searches for tag dictionaries in alternate locations.

Implied tags are automatically assigned to objects by **Smart Tag Rules** in the installed tag dictionaries.

Step 1. Right-click the object whose tags you want to examine and choose **Edit Tags** from the popup menu.

Step 2. To view the **Implied Tags**, select the **Implied Tags** tab in the lower pane of the dialog box.



For example, in the above image, you can see five tags that are implied based on the dictionary rules for a Component object: The first four tags are implied from the Niagara tag dictionary rules:

- n:name
- n:displayName
- n:type
- n:ordInSession

The final tag is implied based on Haystack tag dictionary rules:

- hs:id

Viewing implied tags using Spy view

Implied tags and implied relations are automatically assigned to objects by rules in the installed **Smart Tag Dictionaries**. The implied tags and implied relations do not appear in the **Property Sheet** view or other more commonly used views. **Spy** view shows all of the direct and implied tags and relations on an object as well as other detailed data. Although intended to be used for diagnostic purposes, you can use **Spy** view to identify implied tags and/or relations already assigned to a component. This can be useful when developing hierarchies. Once identified, you can then create queries for those tags/relations in your hierarchy definition.

Prerequisites:

- You are connected to your station.
- One or more installed tag dictionaries. If necessary, add required tag dictionaries to the **TagDictionaryService**.

NOTE: If tagging offline, it is possible that no dictionaries are available. In that situation the system searches for tag dictionaries in alternate locations.

This procedure describes how to open the **Spy** view on a station component to see its implied tags:

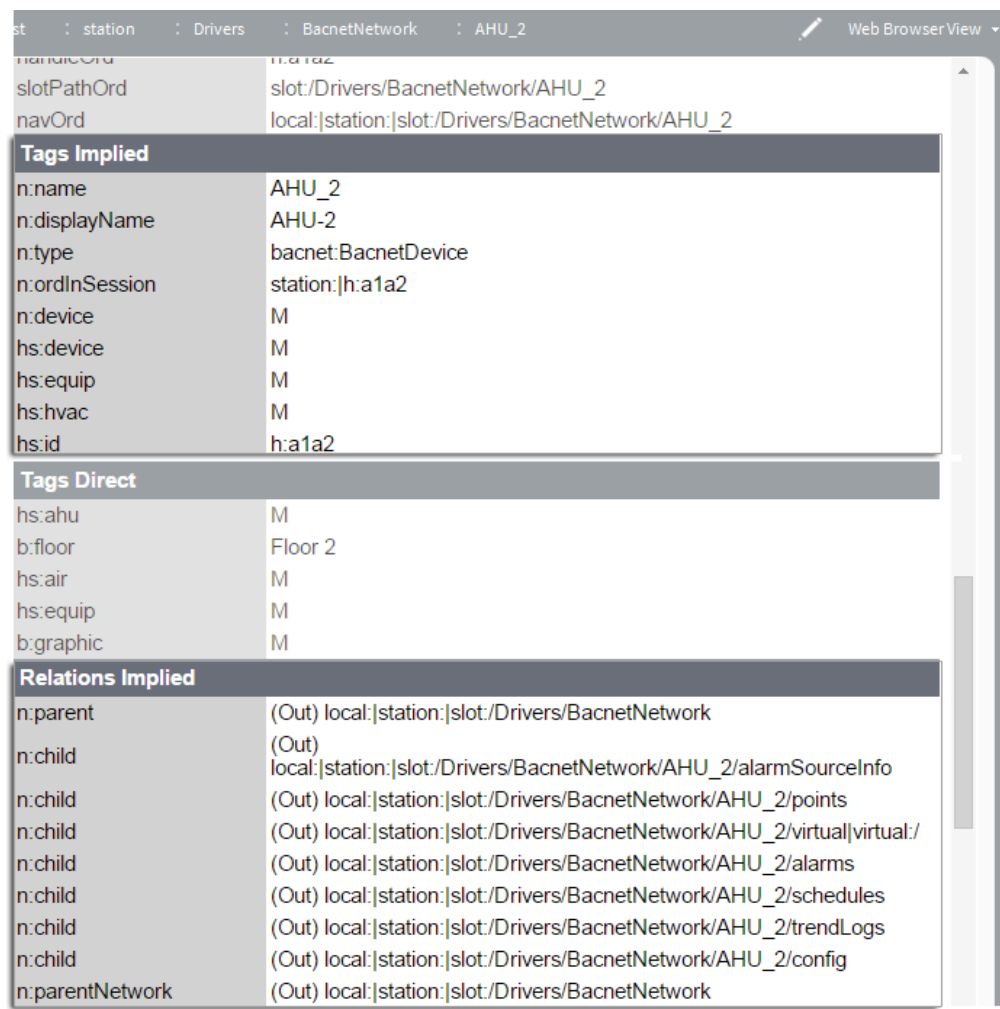
NOTE: Invoking the **Edit Tags** window is another method for viewing the direct and implied tags assigned to a component.

Step 1. In the Nav tree, right-click the component of interest and click **Views > Spy Remote** from the popup menu.
Spy information displays in the **Web Browser View**.

Step 2. Scroll down to **Tags Implied**.

The implied tags assigned to the selected component are listed. Scroll up or down to view all of the tags and relations assigned to the component.

The **Spy** view pictured here lists the different types of tags and relations assigned to the AHU_2 component.




Selecting or exiting tag mode (manager views)

Station Manager and Point Manager views have a Tag Mode available for adding tags to devices or points as they are added.

Prerequisites:

- Tag Mode is only available in the Station Manager or Point Manager views.

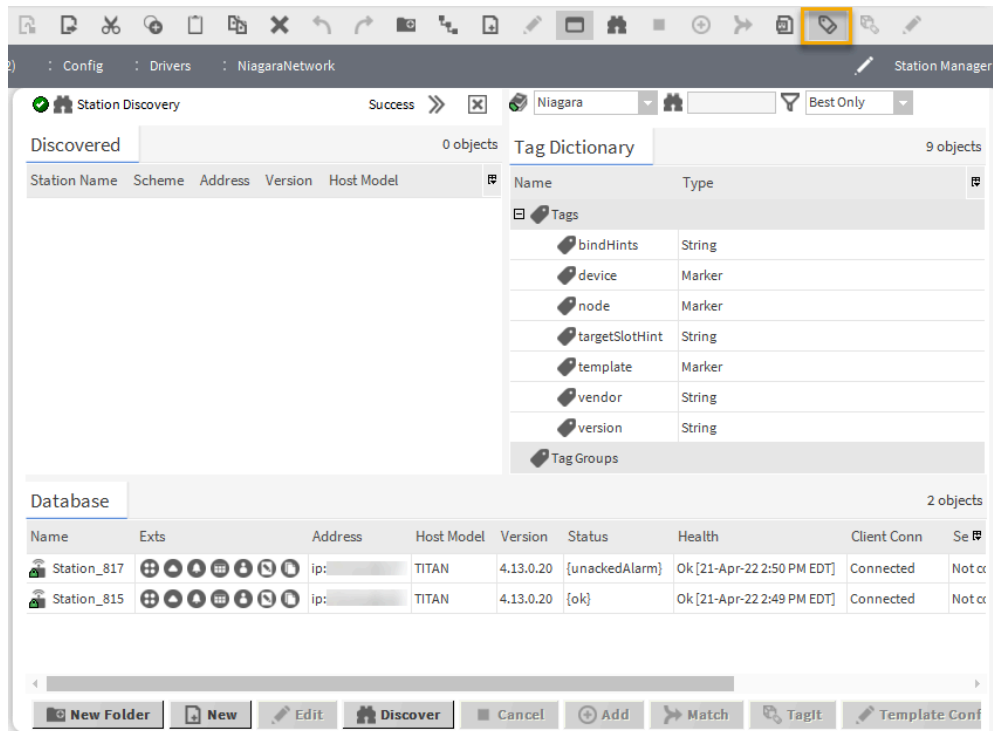
Tagging is integrated into the driver manager views to help you add tags when devices or points are

discovered and added. You can select and exit Tag Mode using either the Manager menu or the  Tag icon in the Toolbar.

While in the Station Manager or Point Manager view, click Manager > Tag Mode from the Workbench main menu to select or exit Tag Mode.

Result

When selected, **Tag Mode** appears as a single pane across the top or as a second pane in the upper pane depending on whether or not you also have **Learn Mode** selected. The following image shows **Tag Mode** and **Learn Mode** selected simultaneously.



Exporting and importing tag dictionaries

The **Tag Dictionary Manager** view provides a method to import and export tag dictionaries (or smart tag dictionaries) in a standard CSV file format, compatible with Excel (or other CSV-compatible spreadsheet software). This facilitates creating custom tag dictionaries, which you then import to your station. Working in the exported CSV file, you can easily edit the correctly structured tag dictionary, populating it with your custom tag definitions, tag rules, etc. When finished, save the revised CSV file and import it using the **Tag Dictionary Manager** view.

Use case

It may suit your purposes to create a custom tag dictionary (or dictionaries) for a specific customer, for an OEM, or for a specific application. You may use a custom tag dictionary as you would other tag dictionaries, to apply tags to objects, create hierarchy definitions, as well as search the station for tagged objects.

NOTE: By default, the license for the Tag Dictionary Service limits the number of tag dictionaries available for the system to the first two. Any dictionaries added above the limit for the license will be in fault and unusable. However, the `Dictionary.Limit` attribute on the license is configurable in the same manner as are device limits.

Creating a new tag dictionary

Creating a new tag dictionary results in a correctly structured, "empty" tag dictionary, which you can export to a CSV file format for editing.

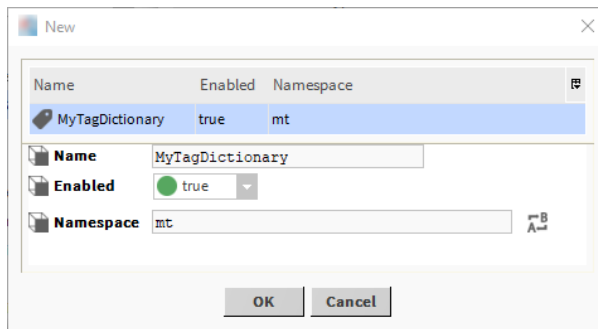
Prerequisites:

You have a license to use tags and have installed the **TagDictionaryService** under the **Services** folder. You have an online or offline connection to an open Station.

Step 1. To open the **Tag Dictionary Manager** view, double-click the **TagDictionaryService** in the Nav tree or

use the right-click menu to open this view.

- Step 2. Click **New**, in the **New** window, select **Smart Tag Dictionary** and click **OK**. A second **New** window opens.



- Step 3. Enter a **Name** and **Namespace** for the dictionary and click **OK**.

Name could be your company name or some other unique name to identify the dictionary.

Namespace should be a short mnemonic to identify the dictionary, similar to the “hs” that stands for Haystack. The shorter, yet meaningful, the better. NEQL predicates require this ID.

The database table of the **Database** pane and the **TagDictionaryService** node in the Nav tree list this custom tag dictionary.

Next steps

Instead of using the **Tag Dictionary Manager**’s **New** button to create a new tag dictionary, you could drag a **SmartTagDictionary** component from the tagdictionary palette to the **TagDictionaryService** in the Nav tree. The **Tag Dictionary Manager** displays the resulting row in `{fault}` because its **Namespace** is not defined. To remove the `{fault}` condition, select the row, click **Edit** and assign a **Namespace**.

Editing a tag dictionary exported to CSV

You can open an exported tag dictionary (CSV format) in Microsoft Excel or other CSV-compatible spreadsheet software, or even a text editor. The file is structured correctly, ready for you to enter a namespace and add tag definitions and other types of definitions, as needed.

Prerequisites:

- Tag dictionary exported to CSV format
- CSV-compatible spreadsheet software

- Step 1. Open the exported CSV file ,which should resemble the one shown here.

	A	B	C	D	E	F	G	H	I	J	K	L
1	namespace											
2							Validity Rules					
3	Section	Rule Name	Group Name	Tag name	Type	Smart Type	hasTags	hasAncestor	isType	hasRelation	hasRelationFilter	Units
4	TagDefinitions											
5	#			name	type	module:class	opt	opt	opt	opt	opt	opt
6	#				AbsTime							
7	#				Boolean							
8	#				Marker							
9	#				Double							
10	#				Float							
11	#				Integer							
12	#				Long							
13	#				Ord							
14	#				RelTime							
15	#				String							
16	#				TimeZone							
17	#				Unit							
18												
19	TagGroupDefinitions											
20	#		groupName				opt	opt	opt	opt	opt	
21												
22	#(tags)			name								
23												
24	RelationDefinitions											
25	#			name	Relation							
26												
27												
28												

- Step 2. In cell B1, enter a **Namespace** for this tag dictionary, typically only a few characters, for example: MyTags .
- NOTE:** Avoid using a namespace that conflicts with that of other installed tag dictionaries, such as "n" (NiagaraTagDictionary) or "hs" (HaystackTagDictionary).
- Step 3. Starting under the **Tag Definition** section, in **Row 18**, enter the first of your tagDefinition entries (one per row). Be be sure to note the following information:

Row	Description
Row 4	Tag Definition section: define the tags for your dictionary in this area.
#	Rows that begin with # are comment lines which show examples or explanatory comments which may be helpful to retain in the file.
Row 5	Row 5 is an example of the pattern to use to define a tagDefinition entry. You must define tag name and tag type.
Row 6–17	Row 6–17 show valid tag types that can be used. Validation Rules columns: Validation rules are optional. These columns are used to define NEQL query predicates that will be used to suggest where the tag can be applied. For a particular tag definition entry, if more than one of these columns have values, they will be wrapped in a tagdictionary:And function. <ul style="list-style-type: none">hasTag: This tag may be applied if the target component also matches this NEQL tag query.hasAncestor: This tag may be applied if the target component has an ancestor that matches this NEQL tag query.

Row	Description
	<ul style="list-style-type: none"> • isType: This tag may be applied to the target component if it is one of these types. Value must be entered in the "module:ClassName" form. If more than one type is entered separated by a space, it will be treated as an "or" function. Example: driver:Device driver:PointFolder: The tag is valid on a BDevice or a BPointFolder. • hasRelation: This tag may be applied if the target component has this relation and has the tags that are listed in the hasRelationFilter (3.c.v) column. hasRelationFilter: This is a tag filter used with hasRelation (3.c.iv) validation check. <p>Units: Units are optional and can be used to define a measurement unit used for a tag that has a value. The value entered is used as the unitName argument in the BUnit.getUnit(String unitName). Example: "square foot" for a tag whose value is an area.</p>

Step 4. Under the TagGroupDefinitions section (optional), add a row for each TagGroup, defining a GroupName that will be used to represent this collection of tags.

- a. Add one or more tag rows under the TagGroup one row for each tag in the group.

NOTE: Tags included in a TagGroup must also be defined in the TagDefinitions section.

Step 5. Under the RelationDefinitions section (optional), add a row for each relation defining a RelationName and enter `Relation` in the Type column.

Step 6. Under the RuleDefinitions section (SmartTagDictionaries only), define the rules for implied tags and relations.

- a. Add a row for each TagRule, enter a RuleName and one or more validation rule column values. See information on Validation Rules listed under Step 3.
- b. Under the tag rule row, add a row for each implied tag for this rule entering a name, type and a smart type. The smart type should be in the module:class format.
- c. If there are any implied tag groups for this rule add a row for each with the GroupName entered in the groupName column.
- d. If there are any implied relations for this tag rule, add a row for each with the RelationName in the name column and enter `Relation` in the type column.

Result

Your edited tag dictionary is complete and ready to import. An example of an edited tag dictionary in CSV file format, shown here.

	A	B	C	D	E	F	G	H	I	J	K	L
1	namespace	my										
2							Validity Rules					
3	Section	Rule Name	Group Name	Tag name	Type	Smart Type	hasTags	hasAncestor	isType	hasRelation	hasRelationFilter	Units
4	TagDefinitions											
5	#			name	type		opt	opt	opt	opt	opt	opt
6	#				AbsTime							
7	#				Boolean							
8	#				Marker							
9	#				Double							
10	#				Float							
11	#				Integer							
12	#				Long							
13	#				Ord							
14	#				RelTime							
15	#				String							
16	#				TimeZone							
17	#				Unit							
18				building	Marker							
19				area	Double	my:building						square foot
20				outside	Marker							
21				air	Marker							
22				temp	Marker							fahrenheit
23	TagGroupDefinitions											
24	#		groupName				opt	opt	opt	opt	opt	
25	#(tags)			name								
26			oaTemp							control:NumericPoint		
27				outside								
28				air								
29				temp								
30	RelationDefinitions											
31	#			name	Relation							
32				buildingRef	Relation							

1

Namespace = my

2

Tag Definitions — notice tagDefinitions, area and temp are configured with Validity Rules

3

Tag Group Definitions — notice oaTemp tagGroup is configured with a Validity Rule and the group contains three tagDefinitions. (outside, air, and temp).

4

Relation Definitions — defines one Relation, buildingRef

Importing a tag dictionary in CSV format

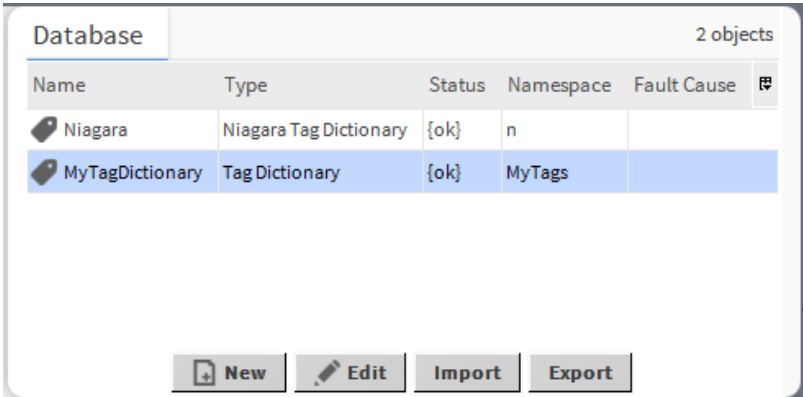
After editing a tag dictionary outside of Niagara this procedure brings the dictionary back into the framework.

Prerequisites:

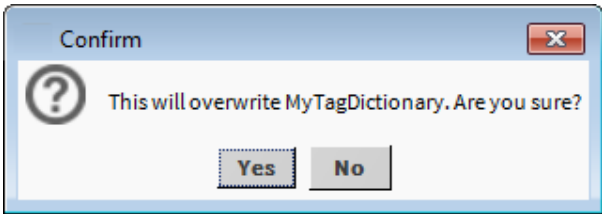
The tag dictionary in CSV format is in your Workbench user home.

Step 1. Open the **Tag Dictionary Manager** view of the **TagDictionaryService**.

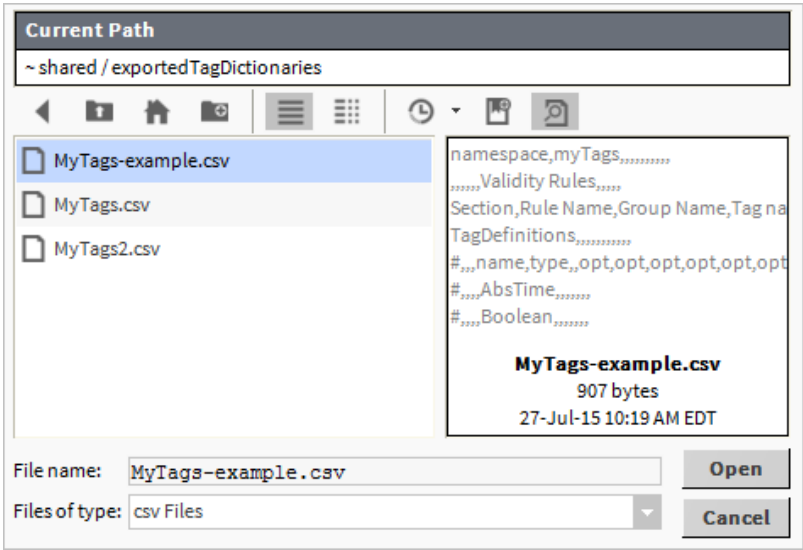
Step 2. Select the tag dictionary to update and click **Import**.



A window confirms that the **Import** action will overwrite the selected tag dictionary.

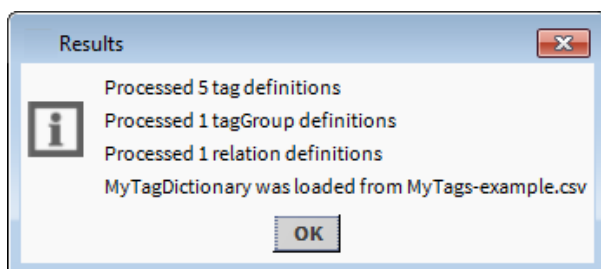


Step 3. To replace the dictionary, click **Yes**.
A **File Chooser** opens.



Step 4. Locate and select the CSV file to import, and click **Open**.
NOTE:
By default, the function prompts for a CSV file. This behavior can be modified programmatically.

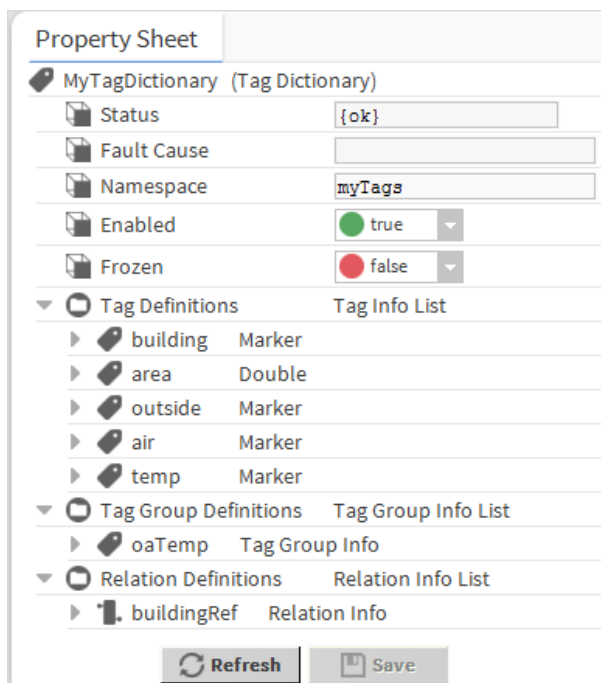
A **Results** window notifies you that the CSV file imported successfully.



Step 5. To continue, click **OK**.

NOTE: In the event that the framework detects an error in the CSV file, an **Error** window opens indicating the error and its location by row or line number.

Step 6. To open the dictionary's **Property Sheet**, expand the **TagDictionaryService** node in the Nav tree and double-click (or right-click) on the imported/updated tag dictionary. The **Property Sheet** opens.



Step 7. Review its properties and verify your changes.

Exporting a tag dictionary

You would export a custom tag dictionary to edit it outside of Niagara. You could export any tag dictionary as an example or as a template.

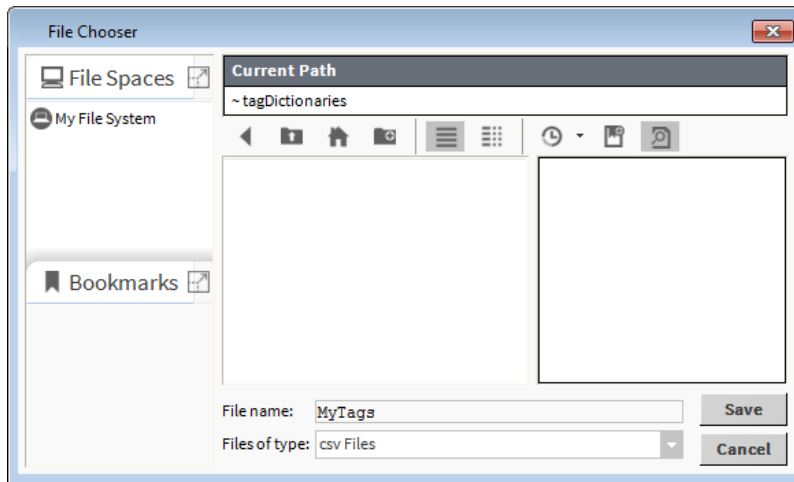
Prerequisites:

You have a license to use tags and have installed the **TagDictionaryService** under the **Services** folder. You have an online or offline connection to an open Station.

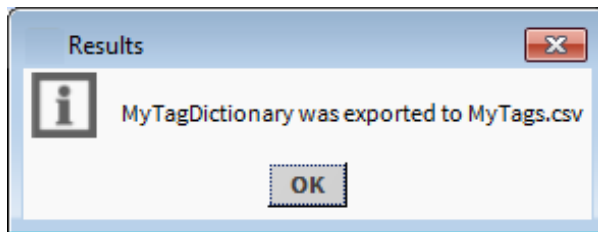
Step 1. To open the **Tag Dictionary Manager** view, double-click the **TagDictionaryService** in the Nav tree or

use the right-click menu to open this view.
The **Tag Dictionary Manager** view opens.

- Step 2. Select a tag dictionary and click **Export**.
The **File Chooser** window opens.



- Step 3. Select a location to save the file, enter the desired file name (as shown), and click **Save**.
A **Results** message confirms the export.



Result

The exported structured tag dictionary is empty at this point. You can edit the file, as well as use it as a template to develop additional tag dictionaries.

NOTE: You can also export the Niagara and Haystack dictionaries to use as examples.

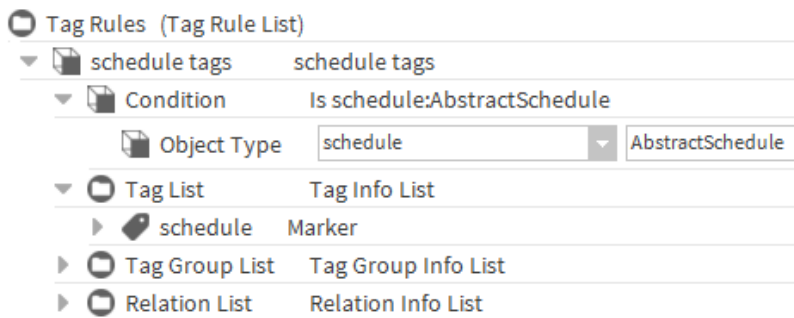
Chapter 3. Tag dictionary service

The **Tag Dictionary Service**, located in a station's **Services** directory, is the container for all tag dictionaries installed in the station.

Smart tag dictionary

The Smart tag dictionary automatically applies the implied tags and relations to objects. Technically, implied items, the implied tags and implied relations, are not added to the station, and the station size is not increased as a consequence. To create a new Smart tag dictionary, drag the SmartTagDictionary component to the **Tag Dictionary Service**. In addition to tag definitions, tag group definitions, and relation definitions present in a simple tag dictionary, a Smart tag dictionary contains a list of tag rules that determine the implied tags and implied relations for each and every object in the station.

Figure 1. Example tag rule



Examples of smart tag dictionaries:

- NiagaraTagDictionary, whose namespace (`<n:>`) is the 'n' character followed by a colon. It is included by default in all stations created using the **New Station Wizard**
- Haystack tag dictionaries, indicated by `<hs:>` namespace (Haystack/Haystack3) or `<h4:>` namespace (Haystack 4). These dictionaries are available from the haystack palette, which is included in the Niagara installation. The Haystack dictionaries are a result of the work of the Haystack community hosted on <http://project-haystack.org>.
- Brick tag dictionary, indicated by `<bk:>` namespace. This dictionary is available from the brick palette, which is included in the Niagara 4.14 and later installations. To learn more about the Brick tag dictionary, see <https://brickschema.org>.

For more details on tag rules see the topic, [tagdictionary-TagRuleList](#).

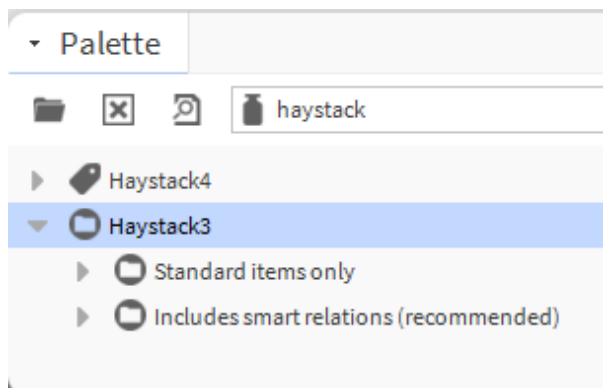
Haystack smart tag dictionary

This module provides the Niagara Haystack smart tag dictionary.

The Haystack dictionary is indicated by the `hs` or `h4` namespace followed by a colon character (:). The haystack module contains the types of components typically present in any smart tag dictionary (tag definitions, tag group definitions, relation definitions, and tag rules). ProjectHaystack.org created and maintains the Haystack tags and equipment point tag groups. As part of that, the organization frequently adds and removes tag definitions. Periodically, a new or updated Niagara release provides the updated haystack module. However, an alternative is available for customers who do not want the updated module or who cannot upgrade software for their whole station.

NOTE: In Niagara, when a Haystack tag dictionary is already installed in a station it is no longer imported automatically when the station starts after upgrading the haystack module. This change prevents any new implied equipRef and siteRef relations from appearing in the station and affecting hierarchies and NEQL results. The latest tag, tag group, and relation definitions can be imported by invoking the **Import Dictionary** action on the HsTagDictionary component.

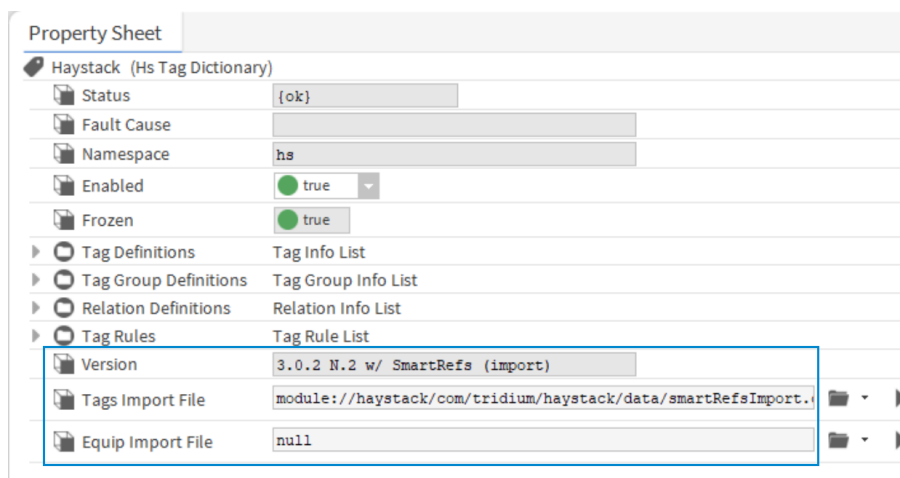
Figure 2. Haystack palette contains 2 versions of the dictionary



In Niagara, the haystack palette provides two versions of the dictionary: one contains only standard items while the other contains the implied equipRef and siteRef smart relations. When adding a new Haystack tag dictionary to a station, the latter dictionary is the recommended one to use to benefit from the smart relations functionality of the dictionary.

The version that includes smart relations has the **Tags Import File** property pre-configured with a file Ord (module://haystack/com/tridium/haystack/data/smartRefsImport.csv) pointing to a file included with the haystack module, as shown here.

Figure 3. Tags Import File field in dictionary that includes smart relations



In latest version of Niagara a jar for the `haystack-rt` module is available, which allows you to modify tag definitions in the haystack dictionary using an external `.csv` file. The patch jar provides two added properties

in the dictionary for this purpose. Using the **Tags Import File** and **Equip Import File** properties, you can point to an external file, or a file within a module, to import tag values overriding those originally in the dictionary.

The installed official haystack dictionary version number is visible in the property sheet. Periodically, the software installation will provide an updated version of the dictionary. Otherwise, if making your own modifications to the dictionary, edit the version number prior to importing your changes. If the version is the same, the import will still accept changes from the TagsImportFile. "(import)" is appended to the version found in the TagsImportFile but that version could be identical to the base version ("3.0.2", for example). For more details on making modifications, see the section on "Modifying the Haystack tag dictionary".

Figure 4. Haystack Tag Dictionary property sheet view

Haystack (Hs Tag Dictionary)	
Status	{ok}
Fault Cause	
Namespace	hs
Enabled	<input checked="" type="checkbox"/> true
Frozen	<input checked="" type="checkbox"/> true
Tag Definitions	Tag Info List
Tag Group Definitions	Tag Group Info List
Relation Definitions	Relation Info List
Tag Rules	Tag Rule List
Version	3.0.2 (import)
Tags Import File	file:^ImportFiles/tagsMerge.csv
Equip Import File	file:^ImportFiles/equipMerge.csv

NOTE: To use these import files, you must always specify a Tags Import File that contains a specified version number even if there are no tags to be modified. If no modifications are made to equipment tagGroups, then the Equip Import File is optional.

Importing changes allows you to modify a tag's type, validity rule, and implied tag rule. In doing this, the tag name is key. You must use the same tag name to override an original tag. You can specify a different value for anything except the original tag name.

You can also add new tag names to add tags to the haystack dictionary, and remove a tag name to remove the tag from the dictionary. To remove a tag, the TagsImportFile must have a row with the tag name and set the value in the Kind column to "Remove". Simply omitting the tag from the TagsImportFile will not remove it from the dictionary. For more details, see the section on "Modifying the Haystack tag dictionary".

Modifying the Haystack tag dictionary

If you wish to create a customized version of the Haystack tag dictionary, you can create haystack import files that you can edit using a CSV-compatible spreadsheet program such as MS Excel. On completion, you can configure the installed tag dictionary to import your changes when the **Import Dictionary** action is manually invoked.

The following procedures describe the steps to create the Haystack tagsImportFile and equipImportFile, edit tags in those import files, and to configure the Haystack dictionary to import your modifications.

For more information on the haystack dictionary in the "HsTagDictionary" topic, see the components section of this guide.

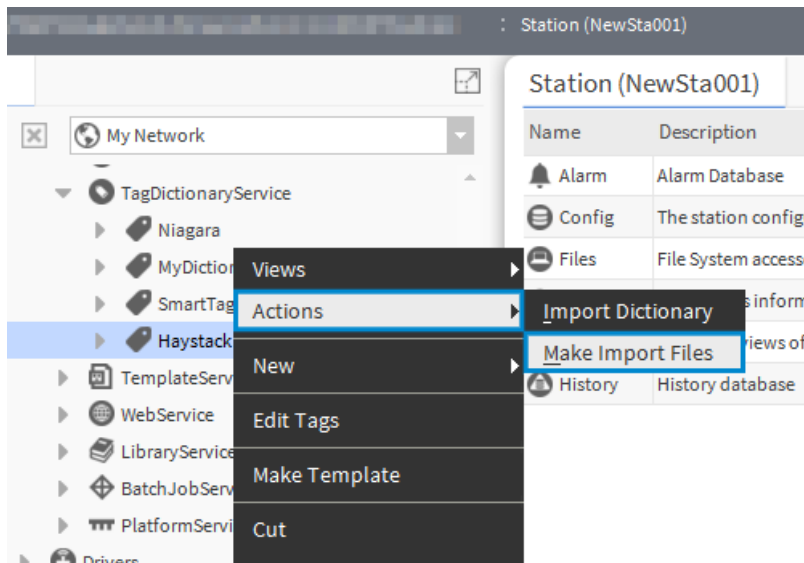
Creating the Haystack tagsImportFile and equipImportFile

The Haystack dictionary has a **Make Import Files** action that can be used to create a `tagsMerge.csv` file and an `equipMerge.csv` file. These files are copies of the master `tags.csv` file from the `haystack-rt` module.

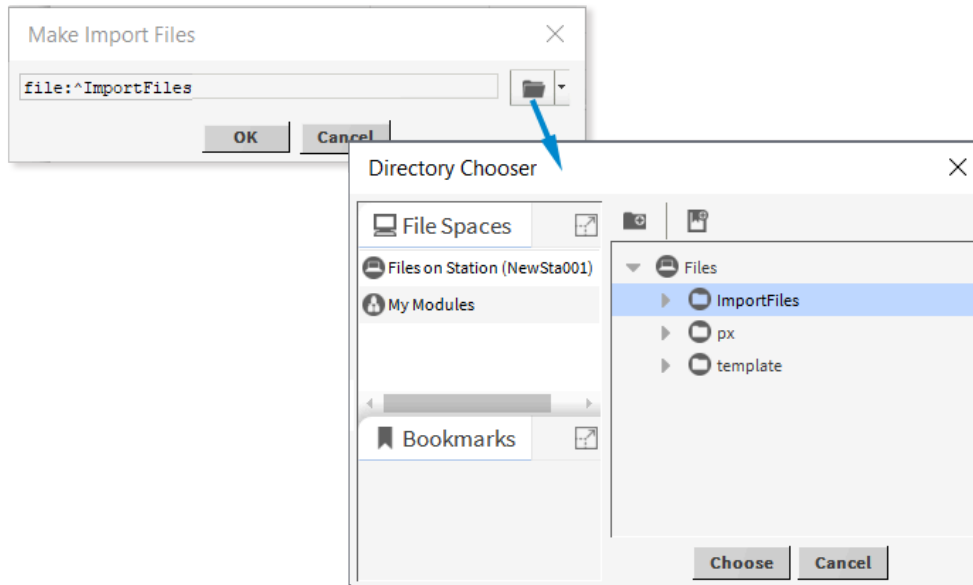
Prerequisites:

- You have an open station connection (local or remote)
- The Haystack tag dictionary is installed in the station.

Step 1. In the NavTree navigate to the station's TagDictionaryService, right-click on the Haystack dictionary and click **Actions > Make Import Files**.



Step 2. In the **Make Import Files** window, click on the **Browse** icon to open a **Directory Chooser** window, navigate to the desired location, select the folder to save the files in, and click **Choose**. In the **Make Import Files** window, click **OK**.



The `tagsMerge.csv` and `equipMerge.csv` files are created and saved to the chosen folder in the station file space.

NOTE: Optionally, if you invoke the action from the Workbench haystack palette with the local station closed, the **Directory Chooser** opens on the local PC file system.

Editing tags in the Haystack `tagsImportFile` and `equipImportFile`

The following steps describe how to remove, modify, and/or add a tag to the file. The example in this procedure describes how to edit the `tagsMerge.csv` file using MS Excel (or another CSV-compatible spreadsheet program).

Prerequisites:

- You have already created the `tagsImportFile` and `equipImportFile`.
NOTE: If the import files are saved to a remote station, you need to transfer them to a PC to edit them with Excel. On the remote platform, use the **File Transfer Client** to copy the files to the local PC file system.
- MS Excel (or other CSV-compatible spreadsheet program) is installed on your PC.

NOTE: Optionally, you can edit the import files using a text editor, but editing is far easier using a spreadsheet program.

Step 1. Navigate to the previously created `tagsMerge.csv` file (or `equipMerge.csv`) and double-click to open it.

Step 2. In the first row of the `tagsMerge.csv` (not the `equipMerge.csv`), edit the **version** value, for example change the number to "3.0.2 MyCompany.1".

NOTE: You must include the version row of the **Tags Import File** (`tagsMerge.csv`). This is true even when all other changes that you make occur in the **Equip Import File** (`equipMerge.csv`).

	A	B	C	
1	version	3.0.2.1	Initial version 10/12/2018	
2				impliedTagRules
3	name	kind	SmartType	hasRelati
4	absorption	Remove		
5	## ac	Marker		
6	## active	Marker		
7	## ahu	Marker		
8	## ahuRef	Ref		

Step 3. Locate the data row containing the tag you wish to remove or modify.

Step 4. For any row that you wish to modify, uncomment the row by removing the "##" prefix from the name column, leaving just the tag name.

NOTE: Initially, all data rows are commented-out (rows start with "##" or "###" characters), and as such they will be ignored on import. To edit a row, you must remove the comment characters so that your changes will be recognized and imported.

Step 5. In the uncommented row, edit any of the tag values as needed. For example, you might make any the following changes:

- To remove the absorption tag, select the tag type, **Marker**, and enter " Remove "
- To change the tag type for ac, active, and air, select the **Marker** value for each and enter a different value, such as " Double ", " String " or " NameTag ".
- To add a tag name to the list, simply insert a row (or copy/paste a row) and enter a unique name for this tag, the tag kind (type), and other values as needed.

NOTE: When making changes, the tag name is important. You must use the exact same tag name to overwrite an existing tag. You can specify a different value for anything except the original tag name. Of course, the exception is when you intend to add a new tag name to the list.

	A	B	C	D	E	F
1				impliedTagRules		
2	name	kind	SmartType	hasTags	hasAncestors	hasRelati
3	absorption	Marker				
4	ac	Marker				
5	active	Marker				
6	ahu	Marker				
7	ahuRef	Ref				
8	air	Marker				
9	airCooled	Marker				
10	angle	Marker				
11	apparent	Marker				

	A	B	C	D	E	F
1						
2	name	kind	SmartType	hasTags	hasAncestors	hasRelati
3	absorption	Remove				
4	ac	Double				
5	active	String				
6	ahu	Marker				
7	ahuRef	Ref			haystack:RefTag	
8	air	NameTag				
9	airCooled	Marker				
10	angle	Marker				

Step 6. When finished making changes, click **File > Save** and **File > Close**.

Result

IMPORTANT: To use the import files you must copy them to a folder on a target station. On a remote platform, use the **File Transfer Client** to copy the files from the PC to the station's "shared" folder.

Configuring the Haystack dictionary to auto-import modifications

Set the Haystack dictionary's **Tags Import File** and **Equip Import File** properties to reference your modified `tagsMerge.csv` and `equipMerge.csv` files and import the changes when you invoke the Haystack dictionary's **Import Dictionary** action.

Prerequisites:

- The Haystack tag dictionary is installed in the station.
- You have already created the **Tags Import File** and **Equip Import File** and modified the files as needed.
- The modified import files are copied to the target station's file space.

Step 1. Open a **Property Sheet** view on the a Haystack dictionary.

Step 2. In the **Tags Import File** field, enter the file Ord for your `tagsMerge.csv` file (use the **Browse** icon to locate and select to the file).

Step 3. In the **Equip Import File** field, enter the file Ord for your `equipMerge.csv` file.

Step 4. Click **Save**

Result

The next time you invoke the Haystack dictionary's **Import Dictionary** action it loads the changes described in the `tagsMerge.csv` and `equipMerge.csv` files.

NOTE: To use these import files, you must always specify a **Tags Import File** that contains a specified version number even if there are no tags to be modified. If no modifications are made to equipment tagGroups, then the **Equip Import File** is optional.

NOTE: When editing `tagsMerge.csv` or `equipMerge.csv`, you must include the version row (in `tagsMerge.csv`). Otherwise, your changes will not be recognized and imported to the dictionary.

haystack-HsTagDictionary

This module provides the Niagara Haystack smart tag dictionary. The Haystack dictionary is indicated by the `hs` namespace followed by a colon character (:). The haystack module contains the types of components typically present in any smart tag dictionary (tag definitions, tag group definitions, relation definitions, and tag rules).

Properties

Besides the standard smart tag dictionary properties, the haystack dictionary contains the following configuration properties.

Name	Value	Description
Version	3.0.2 (default)	Version number for the installed haystack tag dictionary. When the number is appended with "(import)", this indicates that dictionary modifications have been included from the <code>TagsImportFile</code> and/or <code>EquipImportFile</code> .
Tags Import File	null (default)	A file Ord for a CSV file that can be edited using MS Excel (or other spreadsheet program). The file is used to add a new tag or relation, modify an existing tag or relation, or remove an existing tag or relation to/from the haystack dictionary. The file contents are imported to the dictionary when you invoke the

Name	Value	Description
		haystack dictionary's Import Dictionary action.
Equip Import File	null (default)	A file Ord for a CSV file that can be edited using MS Excel (or other spreadsheet program). The file is used to add a new tagGroup, modify an existing tagGroup or remove an existing tagGroup to/from the haystack dictionary. The file contents are imported to the dictionary when you invoke the haystack dictionary's Import Dictionary action.

Actions

- **Import Dictionary** — imports tag, tag group, relation, and tag rule definitions from a standard definition `tags.csv` file and from the optional `TagsImportFile` and `EquipImportFile` files.
- **Make Import Files** — creates example files (`tagsMerge.csv` and `equipMerge.csv`) that can be specified as the `TagsImportFile` and `EquipImportFile` respectively to modify the tags, tag groups, and relations in the installed Haystack tag dictionary.

Haystack Tags Import File format

The **Tags Import File** (`tagsMerge.csv`) is created by the **Make Import Files** action on the Haystack tag dictionary. This file is used to add a new tag or relation, modify an existing tag or relation, or remove an existing tag or relation to/from the haystack dictionary.

The **Tags Import File** is a CSV file that can be edited using MS Excel (or other CSV-compatible software).

IMPORTANT: For editing purposes, the import files can be located anywhere on your PC. However, to use the import files you must copy them to a folder on the target station.

About the rows

Version row: Row 1 — The file must start with "version" in the first column and version string in second column.

Header rows: Rows 2 and 3 — Header rows that contain the column headings.

Data rows: Rows 4 and beyond — Each data row has 12 columns. In a text editor each column is separated with a "," (comma). The column definitions follow. Each row defines an individual tag or relation (Ref kind).

NOTE: Initially, all data rows are commented-out (rows start with "***" or "###" characters), and as such they will be ignored on import. To edit a row, you must remove the comment characters so that your changes will be recognized and imported.

Figure 5. tagsImportFile data in comma-delimited CSV format

```
version,3.0.2,,,,,,,,,
,,,impliedTagRules,,,Validity Rules,,,units
name,kind,SmartType,hasTags,isType,hasAncestors,filter,hasTags,isType,hasAncestor,filter,
## absorption,Marker,,,,,,,,equip chiller,,,,,
```

Figure 6. tagsImportFile data as it appears in MS Excel

	A	B	C	D	E	F	G	H	I	J	K	L
1	version	3.0.2										
2				impliedTagRules				Validity Rules				units
3	name	kind	SmartType	hasTags	isType	asAncestor	filter	hasTags	isType	hasAncesto	filter	
4	## absorption	Marker						equip chiller				

Data columns:

Column	Description
Name	Name of the tag defined by this row. If the tag with this name already exist in the haystack dictionary, the tag’s definition will be overwritten by the definitions in this row. If the tag with this name does not exist in the haystack dictionary, a new tag will be added with this row’s tag definitions.
Kind	Defines the tag kind (e.g. Marker, Bool, Number, Str, URI, Ref, Date, Time, Datetime, Obj, Coord). Note: Ref kind will be defining a relation and not a tag. Note: Remove in the Kind column will cause this tag to be removed from the haystack dictionary.
Smart Type	If the row is defining a smart tag or smart relation this will define the class for the smart type. It is in the form of module:typeName. If no type is entered and there are no implied tag rules, the tag type will be SimpleTagInfo. If no type is entered and there are implied tag rules, the type will be SmartTagInfo If Kind is “Ref” and no type is defined, the relation type will be RelationInfo
impliedTagRules (columns)	Used in defining a TagRule in the TagRules section of the Haystack dictionary. These colums are used to define the condition rules for a smart tag. If more that one column is used then it will use a "and" function to evaluate the combined condition rules. Not used if the Kind is Ref.
ValidityRules (columns)	These columns are used to define the validity for a tag or relation. The validity is used by user interface components to filter tags or relations that may be added to Niagara components. Used in the creating the TagInfo entry for the tag in the TagDefinitions of the Haystack dictionary. NOTE: If the Kind value is “Ref” then it is used in creating the RelationInfo entry in the RelationsDefinitions of the Haystack dictionary.
hasTags	Contains a list of space delimited tags. It will generate a Boolean filter with an "or" between each tag. It is

Column	Description
	indicating that this tag may be implied if the given component has one of these tags. If a haystack tag, only enter the tag name.
isType	Generates a IsTypeCondition using the type provided. In the form of module:typeName. This tag may be implied if the given component is of this type.
hasAncestor	Generates a HasAncestorCondition using the tag specified. If more that one tag is specified then an Or function is used to evaluate. This tag may be implied if the given component has an ancestor with one of these tags. If haystack tag, only enter tag name.
filter	Generates a BooleanFilter using the contents of this filter. The filter must be a fully qualified NEQL predicate. NOTE: You must include nameSpace for tag entries.
Units	Defines the units for the given tag. The value is the unit long name and must be contained in the UnitsDatabase. The value appears as a defaultFacet Facet property added to the TagInfo entry for the tag in the TagDefinitions of the haystack dictionary.

Haystack Equip Import File format

The **Equip Import File** (equipMerge.csv) is created by the **Make Import Files** action on the Haystack tag dictionary. This file is used to add a new tagGroup, modify an existing tagGroup or remove an existing tagGroup to/from the haystack dictionary.

The **Equip Import File** is a CSV file that can be edited using MS Excel (or other CSV-compatible software).

IMPORTANT: For editing purposes, the import files can be located anywhere on your PC. However, to use the import files you must copy them to a folder on the target station.

About the rows

Header rows: Rows 1 and 2 — Header rows that contain the column headings.

Data rows: Rows 3 and beyond — Each row has 12 columns. In a text editor each column is separated with a “,” (comma). The column definitions follow. Each row defines an individual tag or relation (Ref kind).

NOTE: Initially, all data rows are commented-out (rows start with “***” or “##” characters), and as such they will be ignored on import. To edit a row, you must remove the comment characters so that your changes will be recognized and imported.

Figure 7. tagsImportFile data in comma-delimited CSV format

```
,,Validity Rules,,,tagGroup,units
name,Remove,hasTags,isType,hasAncestor,filter,tags,
## activePowerPhaseSensorA,,point,control:ControlPoint,elecPanel,hs:equipRef->hs:elecPanel,active power phase(A) sensor,
```

Figure 8. tagsImportFile data as it appears in MS Excel

	A	B	C	D	E	F	G
1					Validity Rules		tagGroup
2	name	Remove	hasTags	isType	hasAncestor	filter	tags
3	## activePowerPhaseSensorA		point	control:ControlPoint	elecPanel	hs:equipRef->hs:elecPanel	active power phase(A) sensor

Data columns:

Column	Description
Name	Name of the tagGroup defined by this row. If the tagGroup with this name already exists in the haystack dictionary, the tagGroup definition will be overwritten by the definitions in this row. If the tagGroup with this name doesn't exist in the haystack dictionary, a new tag will be added with this row's tag definitions.
Remove	Entering the word "Remove" in this column causes this tagGroup to be removed from the haystack dictionary.
ValidityRules (columns)	This group of columns is used to define the validity for a tagGroup. The validity is used by UI components to filter tagGroups that may be added to a component. Used in creating the TagGroupInfo entry for the tagGroup in the TagGroupDefinitions of the Haystack dictionary.
hasTags	Contains a list of space delimited tags. It generates a Boolean filter with an "Or" between each tag. It indicates that this tagGroup may be valid to be applied to the given component if it has one of these tags. If haystack tag, only enter the tag name.
isType	Generates the IsTypeCondition using the type provided. Written in the form of <code>module:typeName</code> . This tagGroup may be valid to be applied to the given component is of this type.
hasAncestor	Generates a HasAncestorCondition using the tag(s) specified. If more than one tag is specified then an <code>Or</code> function will be used to evaluate. This tag may be valid to be applied to the given component if it has an ancestor with one of these tags. If it is a haystack tag, only enter the tag name.
filter	Generates a BooleanFilter using the contents of this filter. The filter must be a fully qualified NEQL predicate. It is typically used to traverse a <code>hs:equipRef</code> relation to validate that this point is related to a specific type of equipment. NOTE: You must include the namespace for tag and relation entries.
TagGroup Tags	Contains a space delimited list of discrete tags that are associated with this tagGroup. Most of these tags will be Marker tags. If a tag is a value tag, a value can be specified by using <code>tagName(tagValue)</code> format. Example: <code>cool stage(2) cmd</code> specifies tagGroup tags : <code>cool stage cmd</code> with the <code>stage</code> tag having a value of <code>2</code> .

haystack-EquipRelation

Haystack tag dictionary is the addition of the smart equipment relation (`hs:equipRef`).

Typically you create an explicit direct relation between all points that belong to an equipment object and the component in the station that represents the equipment. Instead, you can use this smart equip type relation to automatically create an implied relation between a point and its ancestor if that ancestor has the `hs:equip` tag applied.

The smart EquipRef relation implies an `hs:equipRef` between a Control Point and the nearest ancestor with the `hs:equip` tag. This relation is not implied on a Control Point whose proxy extension is the `NullProxyExt`. This relation is not implied on a Control Point if the point already has a direct `hs:equipRef` relation.

haystack-SiteRelation

Haystack tag dictionary is the addition of the smart site relation (`hs:siteRef`).

The haystack module supports a smart site relation (`hs:siteRef`) that is valid for ControlPoints. If a ControlPoint has an `hs:equipRef` relation (direct or implied) to a component with the `hs:equip` tag and that equip component has an `hs:siteRef` relation to a component with the `hs:site` tag, an `hs:siteRef` relation is implied from the ControlPoint to that site component.

H4TagDictionary

With the Haystack 4 tag dictionary, most items come directly from the unaltered Project Haystack source files (see <https://project-haystack.org/download>). This provides the benefit that it does not require post-processing of Project Haystack sources to supply information necessary to Niagara. The latest version of these source files at the time of the release of Niagara is packaged with the haystack-rt module. Newer versions that were released later by Project Haystack may be used instead of the packaged versions.

As of Niagara 4.13, the Haystack 4 tag dictionary primarily uses `defs.json` and `protos.json` files produced by Project Haystack to generate the dictionary's tags, tag groups, relations, and tag rules.

`Defs.json` contains information about all tags, all relations, and tag groups based on conjuncts. It also contains the tag hierarchy that is used to create most tag rules.

`Protos.json` contains additional tag groups.

An additional configuration file is used to assist with the dictionary generation, and add tag rules as well as smart tag types that enable some tagging convenience features. A default file is packaged with `haystack-rt`, but you can reference your own, if required. For stations already tagged in Haystack 3, a migration action adds Haystack 4 items that are equivalent to the Haystack 3 versions.

The Haystack 4 tag dictionary's namespace is `h4`.

Migrating to Haystack 4

The Haystack 4 tag dictionary contains an action that adds Haystack 4 equivalents for Haystack 3 tags, tag groups, and relations. The Haystack 3 items are not removed. The equivalent tags and relations are mostly copies of the Haystack 3 versions with the distinction of the Haystack 4 `h4` namespace. To see the required item modifications, see "Changes3to4 – Project Haystack" at <https://project-haystack.org/doc/docHaystack/Changes3to4>.

Tag groups

These modifications are captured in a configuration file packaged with the `haystack-rt` module.

There are no exact equivalents for Haystack 3 tag groups. Some require only an additional "point" tag such as the Haystack 3 `dischargeAirTempSensor` group that results in adding the Haystack 4 proto `dischargeAirTempSensorPoint`. Others require an explicit mapping such as `energyNetSensor` to `totalNetAcElecActiveEnergySensorPoint`. Some Haystack 3 tag groups, such as `steamEnteringFlowSensor`, cannot be mapped and Haystack 4 equivalent tags are added for those tag group tags.

Value tags are not included in Haystack 4 tag groups. Therefore as an example, the `voltAnglePhaseSensorAB` tag is equivalent to the `acElecVoltAngleSensorPoint` tag plus a "phase" tag set to "AB".

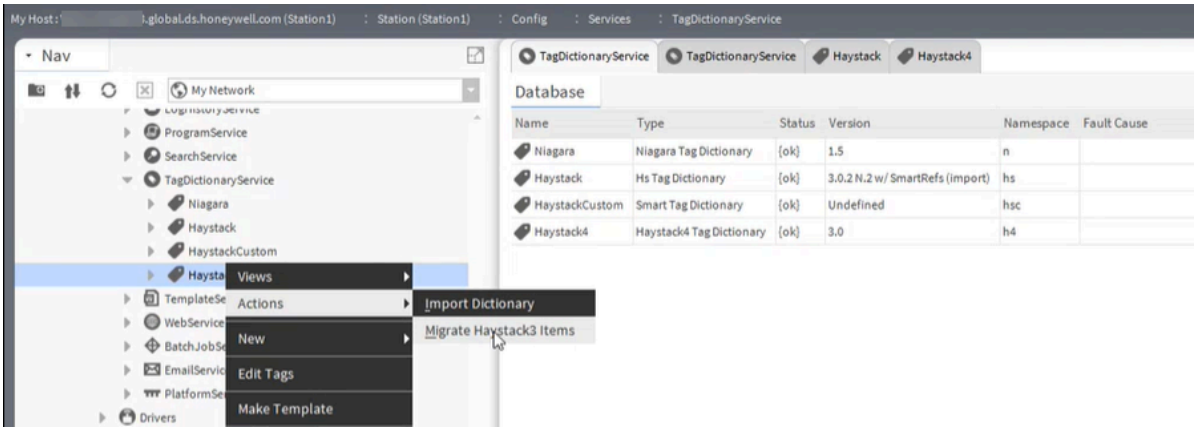
Migrating Haystack 3 items

For stations already tagged in Haystack 3, a migration action adds Haystack 4 items that are equivalent to the Haystack 3 versions.

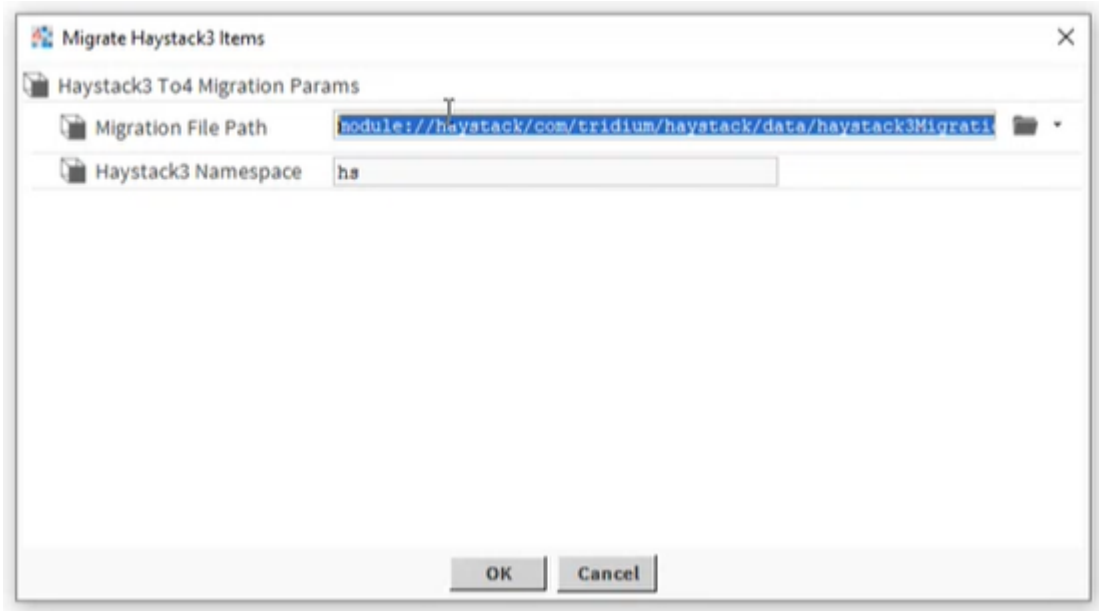
Prerequisites:

- You have an open station connection (local or remote).
- The Haystack 4 tag dictionary is installed.

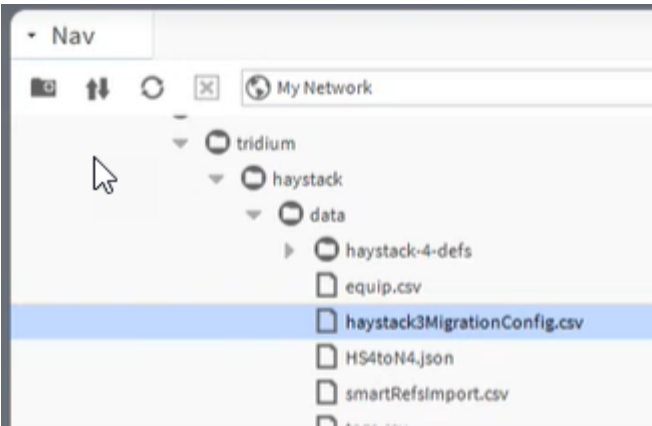
Step 1. To invoke the migration action, under `TagDictionaryService`, right-click `Haystack4` and select **Actions > Migrate Haystack3 Items**.



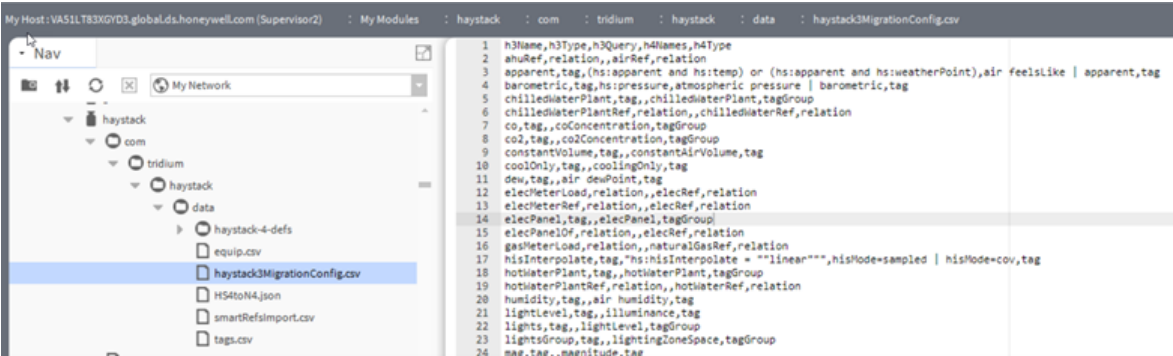
The **Migrate Haystack3 Items** window opens.



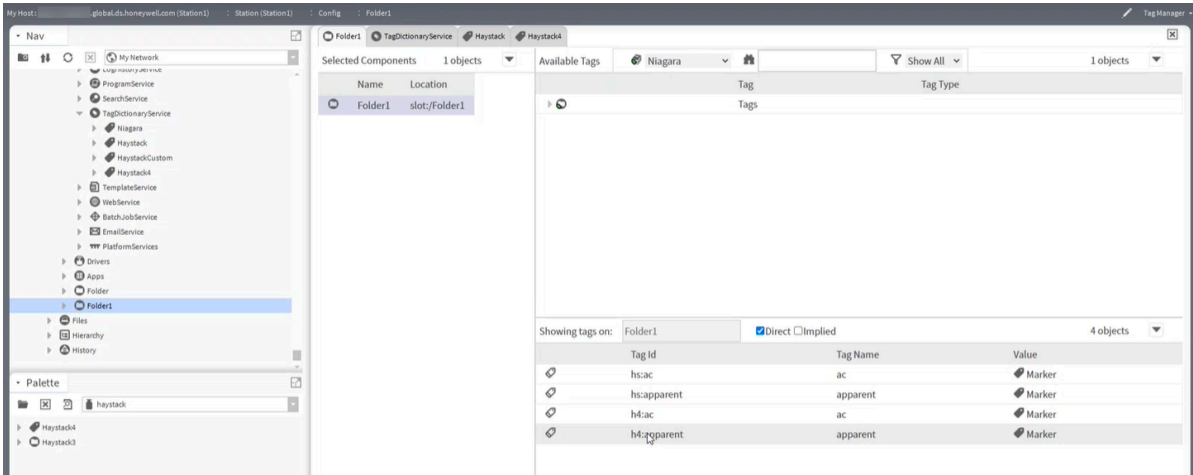
The migration action accepts a customized namespace. By default the `module://haystack/com/tridium/haystack/data/haystack3MigrationConfig.csv` configuration file path is selected. You can find the config file under the Haystack module.



If desired, you can make changes to the configuration file. The configuration file consists of the following columns: haystack3 name, type (tag, tagGroup, or relation), optional NEQL query, corresponding h4 name(s), and h4 type (tag, tagGroup, or relation).

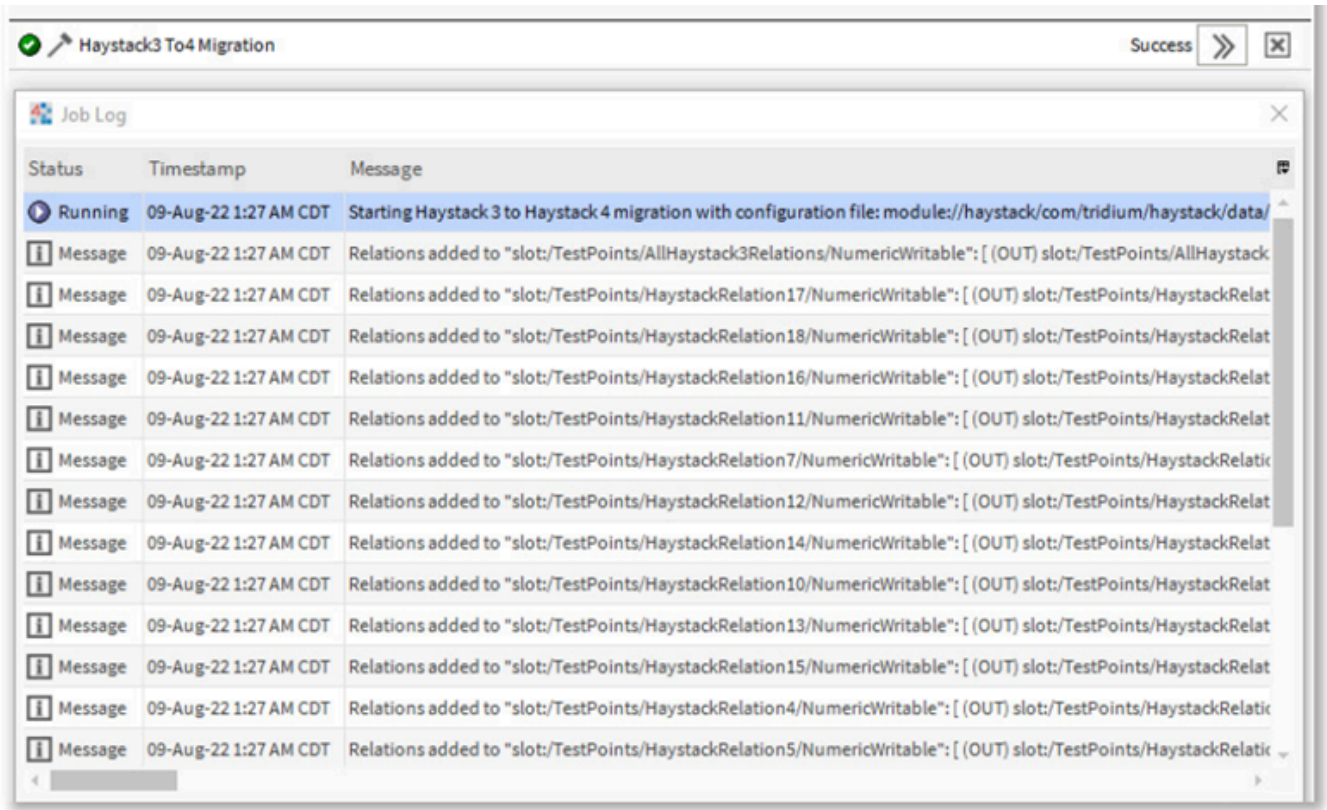


Step 2. Click OK to execute the migration action.



Result

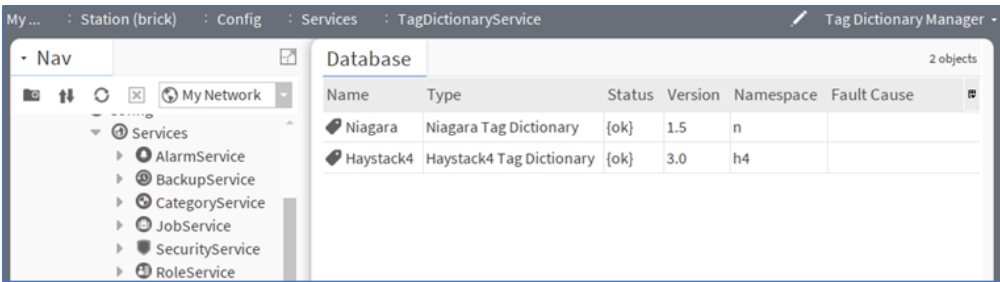
The migration action kicks off a job that logs the actions taken on each component in the station. Any errors are also logged. Once you take action to correct these errors, they can re-run the job as required until the migration is complete.



Haystack 4 import

For the Haystack 4 tag dictionary, the haystack-rt.jar is packaged with the latest versions of Project Haystack source at the time of the Niagara release. There is a small set of items defined in a Niagara configuration file that you can customize, if necessary.

Figure 9. The Tag Dictionary Service with the Haystack 4 dictionary



The following sections give you a detailed overview of the Haystack 4 tag dictionary import.

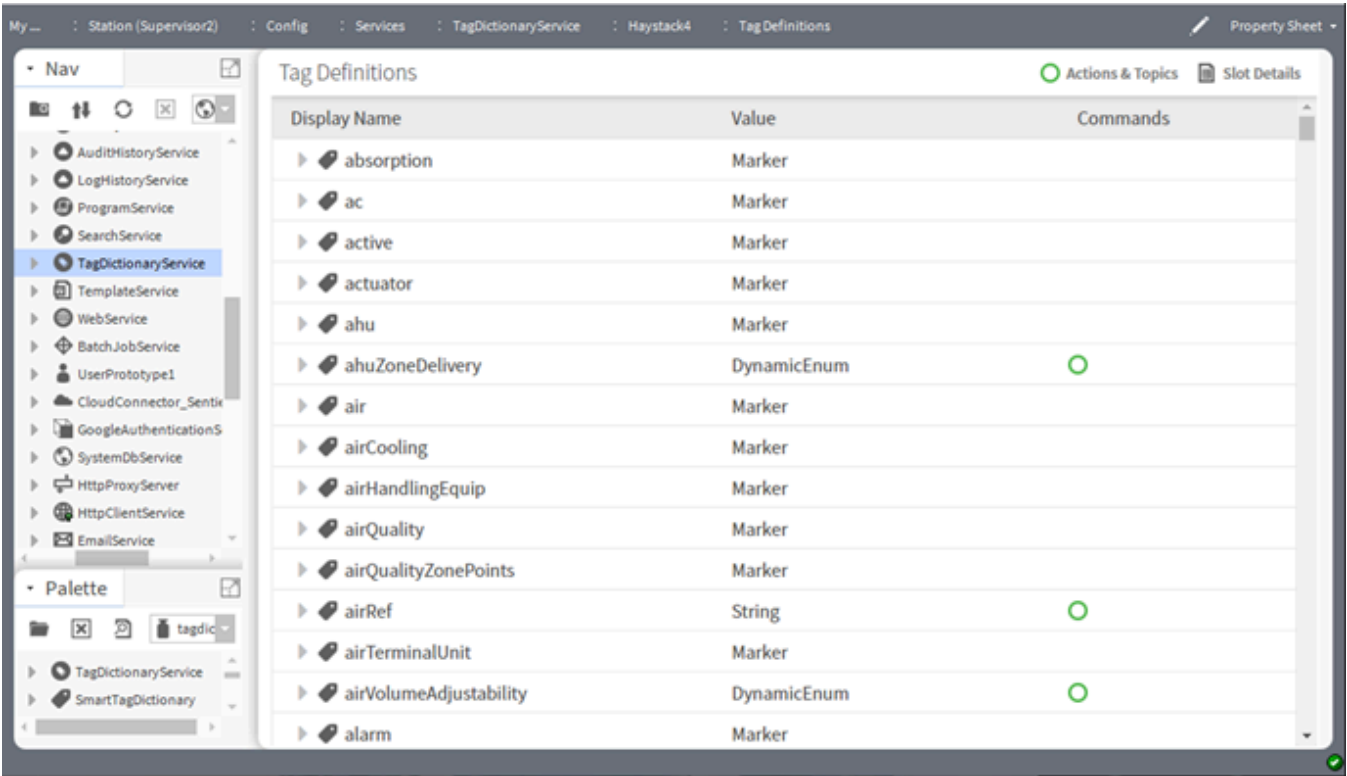
Tags

All tags in the Niagara Haystack 4 tag dictionary come from the Project Haystack’s `defs.json` file. Most defs in the Project Haystack core library (lib:ph) are excluded unless they meet the following requirements:

- The entity or `geoPlace` defs
- Subtypes of entity
- Defs that are a tag on entity or `geoPlace` or one of their subtypes
- Defs listed in the Niagara configuration file: min, max, input, and output

For remaining defs, the supertype tree of each def is traversed and if a mapped Haystack 4 type is found, a tag of the corresponding Niagara type will be added to the dictionary.

Figure 10. Haystack 4 tag definitions



Choice tags

All of the choice values for a choice def subtype are added as separate tags in the dictionary.

If a choice def has an “of” value, such as `pipeFluid` where “of” is the fluid def, the choice values are all descendants of that “of” def. Otherwise, the choice values are simply the descendants of the choice def.

Figure 11. Haystack 4 choice tags

Selected Components

Name	Location
Pipe	slot:/Pipe

Available Tags

Haystack4

pipe

Show All

Tag	Tag Type
Tags	
pipe	Marker
pipeFluid	DynamicEnum
pipeSection	DynamicEnum
Tag Groups	

Showing tags on: Pipe ☒ Direct ☒ Implied 13 objects

Tag Id	Tag Name	Value
h4:pipeFluid	pipeFluid	water
h4:dis	dis	Pipe
n:displayName	displayName	Pipe
h4:phenomenon	phenomenon	Marker
h4:water	water	Marker
h4:liquid	liquid	Marker

Remove Components

Edit

Add

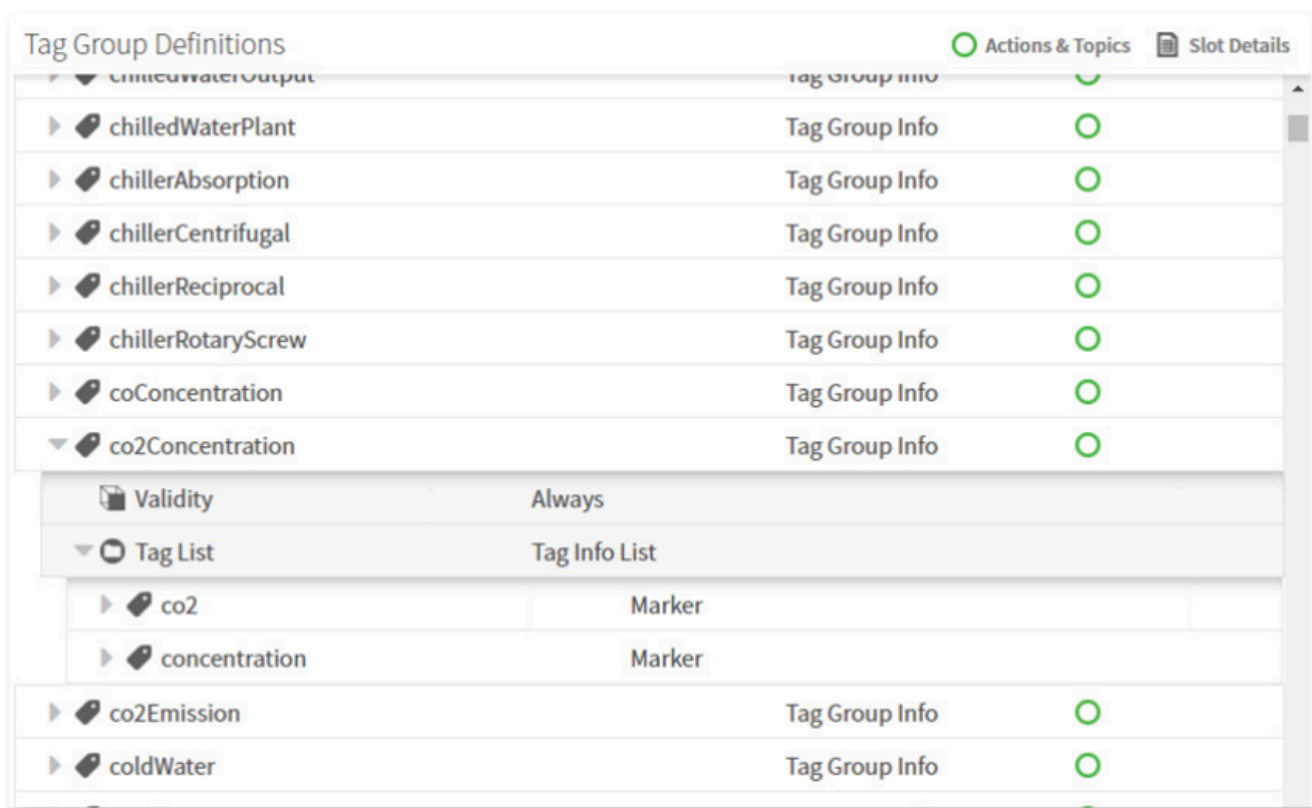
Delete Tags

The choice def is added as a BDynamicEnum tag with the choice values included in the value’s BEnumRange. Tag rules are added to imply the corresponding choice value tag based on the selected enum value.

For example, if the pipeFluid tag on a component is set to “water,” a water marker tag will be implied on that component.

Tag group

The first set of tag groups in the Niagara Haystack 4 tag dictionary are derived from the conjuncts (for example, ac-elec) defined in the defs.json file.
























Figure 12. Example of conjunct tag groups

The screenshot shows the 'Tag Group Definitions' window with two tabs: 'Actions & Topics' (selected) and 'Slot Details'. The main table lists tag groups, each with a tag icon, a name, a 'Tag Group Info' link, and a status indicator (a green circle). The 'co2Concentration' group is expanded, showing its 'Validity' as 'Always' and its 'Tag List' containing 'co2' and 'concentration', both marked as 'Marker'.

Tag Group Definitions		Actions & Topics	Slot Details
chilledWaterOutput	Tag Group Info	○	
chilledWaterPlant	Tag Group Info	○	
chillerAbsorption	Tag Group Info	○	
chillerCentrifugal	Tag Group Info	○	
chillerReciprocal	Tag Group Info	○	
chillerRotaryScrew	Tag Group Info	○	
coConcentration	Tag Group Info	○	
co2Concentration	Tag Group Info	○	
Validity Always			
Tag List Tag Info List			
co2	Marker		
concentration	Marker		
co2Emission	Tag Group Info	○	
coldWater	Tag Group Info	○	

The co2-concentration conjunct becomes a co2Concentration tag group that contains the co2 and concentration tags.

Figure 13. Example of protos tag group

Tag Group Definitions			 Actions & Topics	 Slot Details
 acElecVoltImbalanceSensorPoint	Tag Group Info			
 acElecVoltMagnitudeSensorPoint	Tag Group Info			
 acElecVoltThdSensorPoint	Tag Group Info			
 activeAcElecPowerSensorPoint	Tag Group Info			
 airCo2ConcentrationSensorPoint	Tag Group Info			
 Validity	Always			
 Tag List	Tag Info List			
 air	Marker			
 co2	Marker			
 concentration	Marker			
 sensor	Marker			
 point	Marker			
 airCo2ConcentrationSpPoint	Tag Group Info			
 airDewPointSensorPoint	Tag Group Info			

The next set of tag groups are created for all protos in Project Haystack’s `protos.json` file that are not already a tag or conjunct. For example, the “point” and “humidifier equip” protos are skipped because there is already a point tag and humidifierEquip conjunct tag group.

Relations

For all defs that are a subtype of the ref def, a string value tag is added.

You can add these tags directly to station components and set their value manually to the ID of the entity to which they refer. If the ID of that referenced entity changes, the values of these ref tags must be manually updated. Alternatively, Niagara relations can be used. For ref def subtypes except for ID, a relation is added to the dictionary. These relations are exported by the `nhaystack` service as ref tags with their value set to the ID of the relation endpoint.

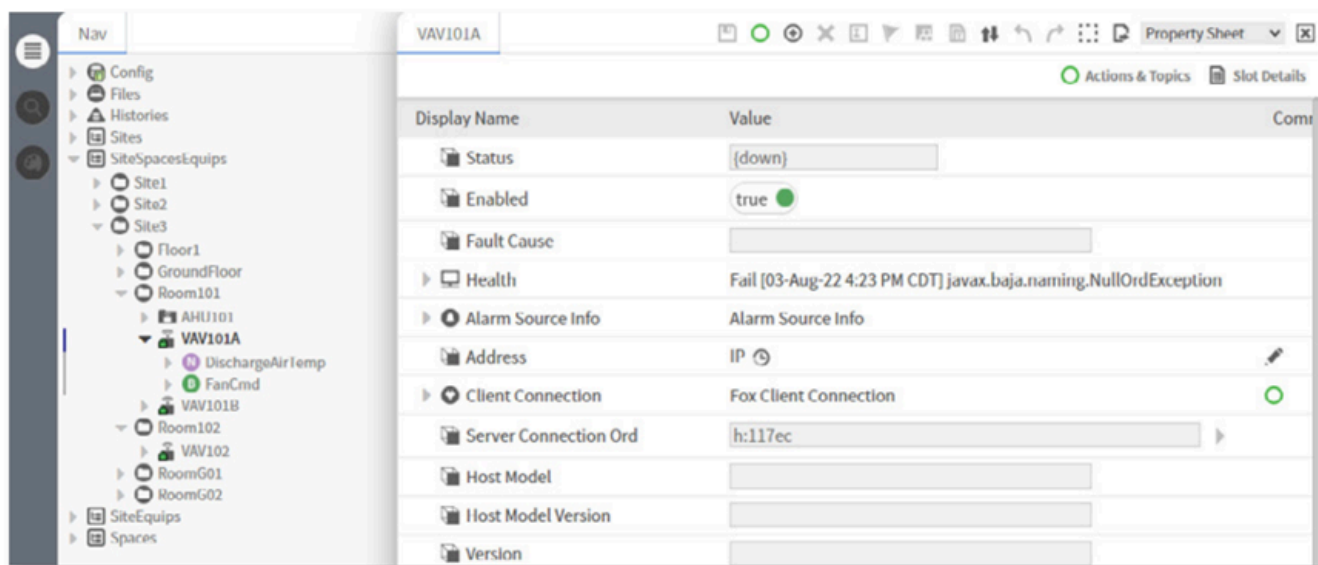
Display Name	Value	Commands
airRef	Relation Info	
blowdownWaterRef	Relation Info	
chilledWaterRef	Relation Info	
condensateRef	Relation Info	
condenserWaterRef	Relation Info	
deviceRef	Relation Info	
domesticWaterRef	Relation Info	
elecRef	Relation Info	
equipRef	Relation Info	
fuelOilRef	Relation Info	
gasolineRef	Relation Info	
hotWaterRef	Relation Info	
makeupWaterRef	Relation Info	
naturalGasRef	Relation Info	
networkRef	Relation Info	
refrigerantRef	Relation Info	

The Niagara configuration file also defines tag rules that imply smart `equipRef`, `spaceRef`, and `siteRef` relations. The smart `equipRef` relation works exactly as the Haystack 3 version, which is that an outbound relation is implied from non-null proxy points to a component ancestor with the `equip` tag and inbound relations are implied from that ancestor back to the points.

Unlike Tridium's Haystack 3 tag dictionary, the `equip` tag is not implied on all BDevices, so it needs to be added where appropriate.

If an **equip** component has a `spaceRef` relation to a **space** component, the smart `spaceRef` relation will imply outbound `spaceRef` relations from that **equip**'s points to that **space** and will imply inbound relations from the **space** component back to those points.

If an **equip** or **space** component has a `siteRef` relation to a **site** component, the smart `siteRef` relation will imply outbound `siteRef` relations from the sub-equip, sub-spaces, and equip points to that **site** and will imply inbound relations from the **site** component back to those items. The following figure shows an automatically generated hierarchy based upon these implied smart relations.



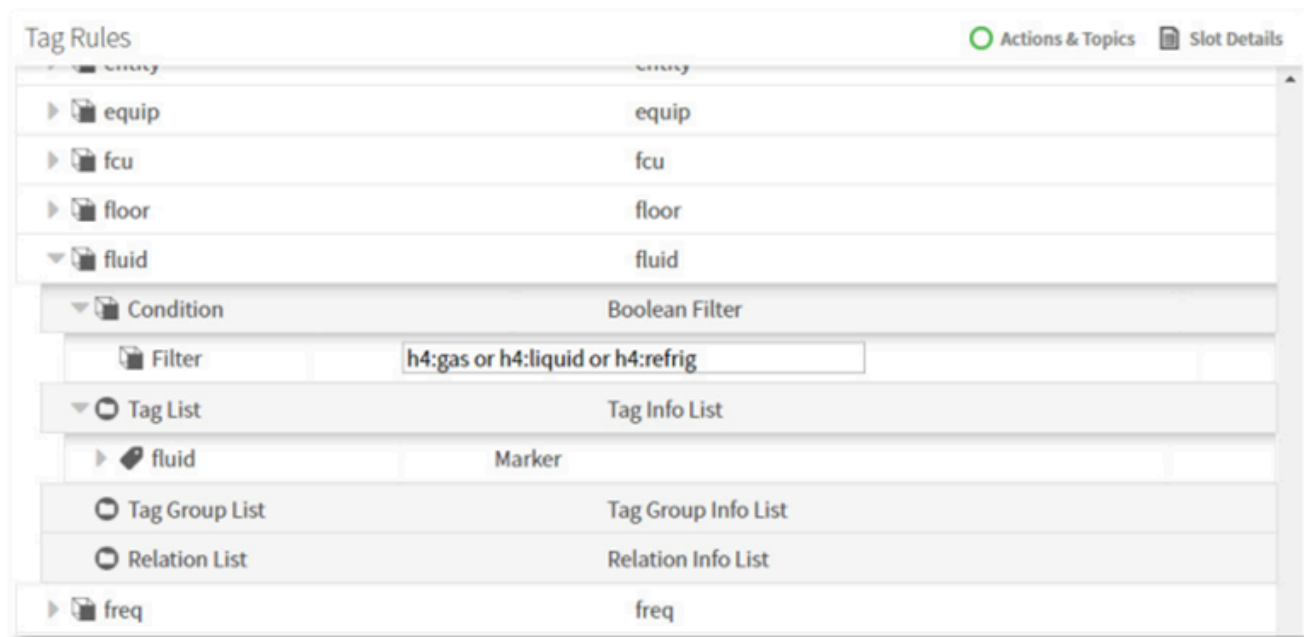
Tag rules

The following gives you an overview of standard and custom tag rules.

Standard

Rules for the choice value tags are automatically generated based on the choice values found in the `defs.json` file. Another set of rules is automatically generated based on the def type inheritance tree in Haystack 4.

For example, water is a subtype of liquid, which is a subtype of fluid, which is a subtype of substance, which is a subtype of phenomenon. As a result, there is a tag rule that implies the liquid tag if a component has the water tag, or a tag rule that implies the fluid tag if a component has the liquid tag (direct or implied).

Figure 14. Inheritance tree tag rules

Custom

The Niagara configuration file defines tag rules that unlock some tagging convenience features in the framework. There are rules that imply simple tags based on the component's type such as the point tag on BControlPoints and the bacnet tag on BBacnetNetworks. There are rules implying smart tags that derive their value from something else such as the unit tag whose value is based on the component's units facet. An alternate configuration file can be specified if you wish a different set of tag rules and/or if you want to change the tag types they imply.

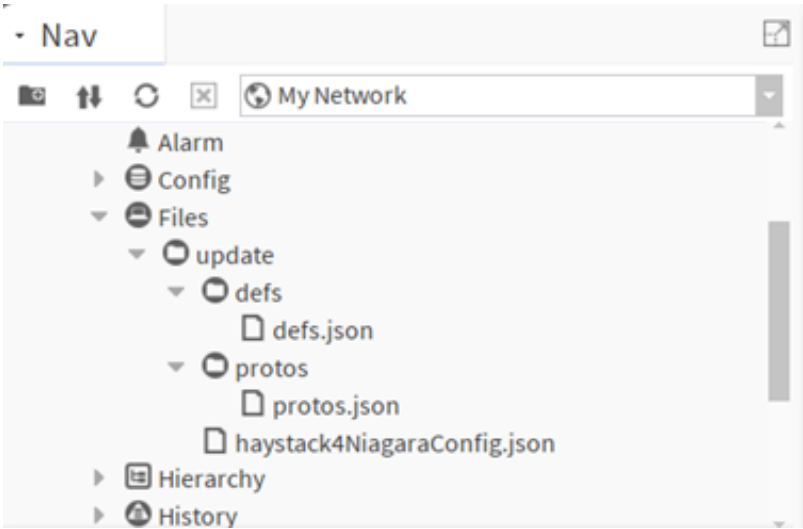
Figure 15. Custom tag rules

Tag Rules	
▶ 📁 dis	dis
▶ 📁 equipRef	equipRef
▶ 📁 his	his
▶ 📁 hisErr	hisErr
▶ 📁 hisMode	hisMode
▶ 📁 hisStatus	hisStatus
▶ 📁 id	id
▶ 📁 kind	kind
▶ 📁 maxVal	maxVal
▶ 📁 minVal	minVal
▶ 📁 network	network

Updating an existing H4 tag dictionary

You can update an existing Haystack 4 tag dictionary on a running station with each tag dictionary update that Haystack releases.

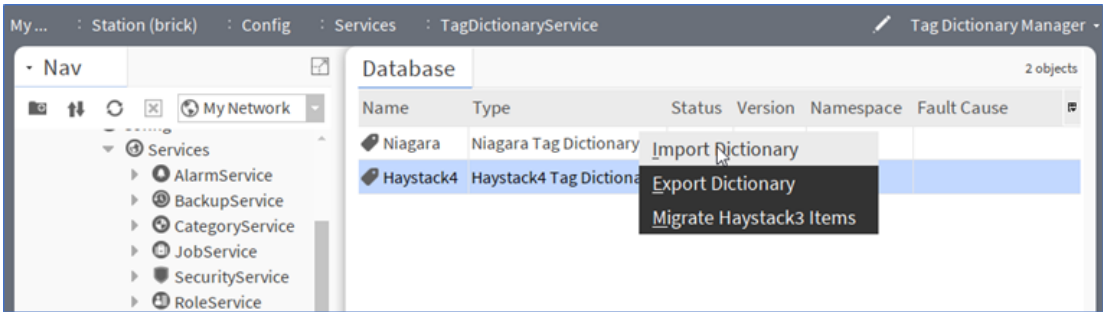
- Step 1. Go to the "Project Haystack" website at <https://project-haystack.org/download> and download the definitions and prototypes.



Move the `defs.json` and `protos.json` files into a folder in the station's shared folder. Move the `defs.json` file into a `defs` folder and the `protos.json` into a `protos` folder.

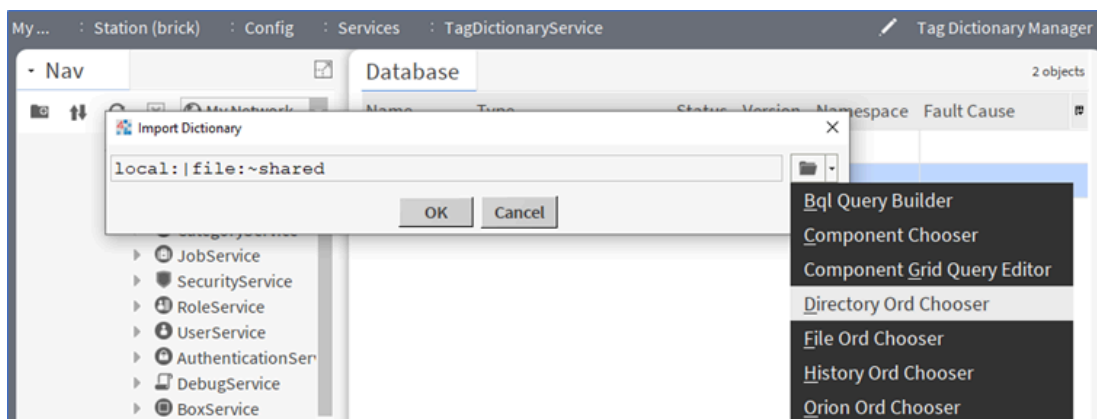
NOTE: If you have customized your Haystack dictionary tag rules, for example, removed a default tag rule, the configurations for those customizations are contained in the third file, which is named `haystack4NiagaraConfig.json`. If no customization is done, the default tag rule configuration is used. You can find the default tag rule configuration file `HS4toN4.json` in the `haystack-rt` module.

- Step 2. To perform the tag dictionary update, right-click on the Haystack dictionary in the Tag Dictionary Manager view and select **Import Dictionary** from the drop-down menu.



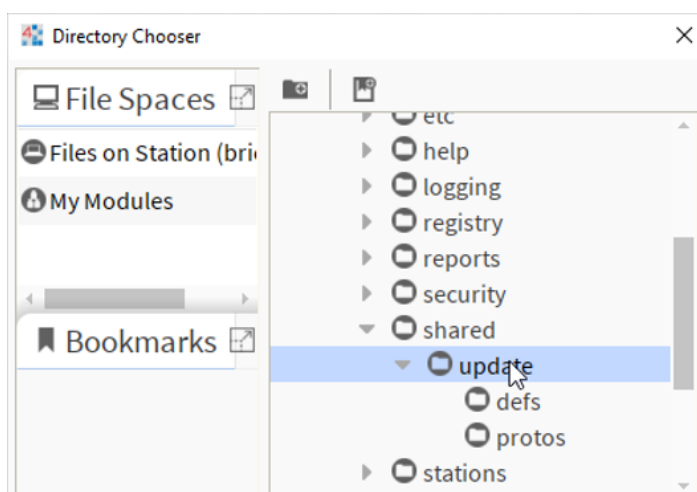
The **Import Dictionary** window opens.

- Step 3. In the upper right corner of the **Import Dictionary** window, click the down arrow and select **Dictionary Ord Chooser** from the drop-down menu.



Step 4. To run the import, navigate to the files in the station, select the folders containing the `defs.json` and `protos.json` files, and click **OK**.

Step 5. In the **Directory Chooser** window, select the `update` folder located in the station.



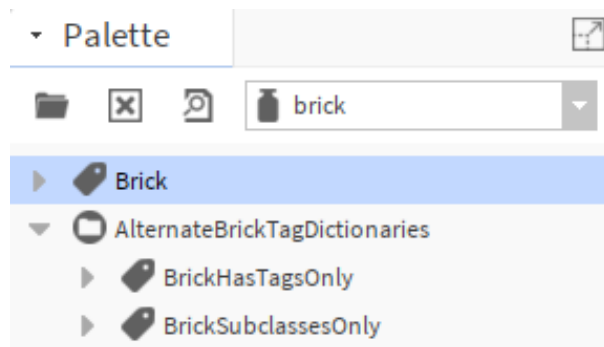
Step 6. After import completion, you can open the directory entry in the property sheet view to confirm that the import was successful and that the tags, groups, relations, and rules are updated as intended.

Brick tag dictionary

The Brick tag dictionary is an instance of the standard Niagara Smart Tag Dictionary and does not contain any custom properties or actions. There are no custom views associated with this dictionary. The dictionary contains a collection of definitions for tags, tag groups, relations, and tag rules.

To create a Brick tag dictionary, establish a Fox connection to the station. In the brick palette, select the **Brick** component and add it to the **TagDictionaryService** container on the station.

Starting with Niagara 4.15, you have three tag dictionary components available for installation from the brick palette:



- The main **Brick** tag dictionary includes tag groups that contain tags derived from the Brick schema subclass and **hasAssociatedTag** information.
- The **BrickHasTagsOnly** dictionary in the **AlternateBrickTagDictionaries** folder contains tag groups with tags derived only from the **hasAssociatedTag** information .
- The **BrickSubclassesOnly** dictionary in the **AlternateBrickTagDictionaries** folder contains tag groups with tags derived only from subclass information.

Overview

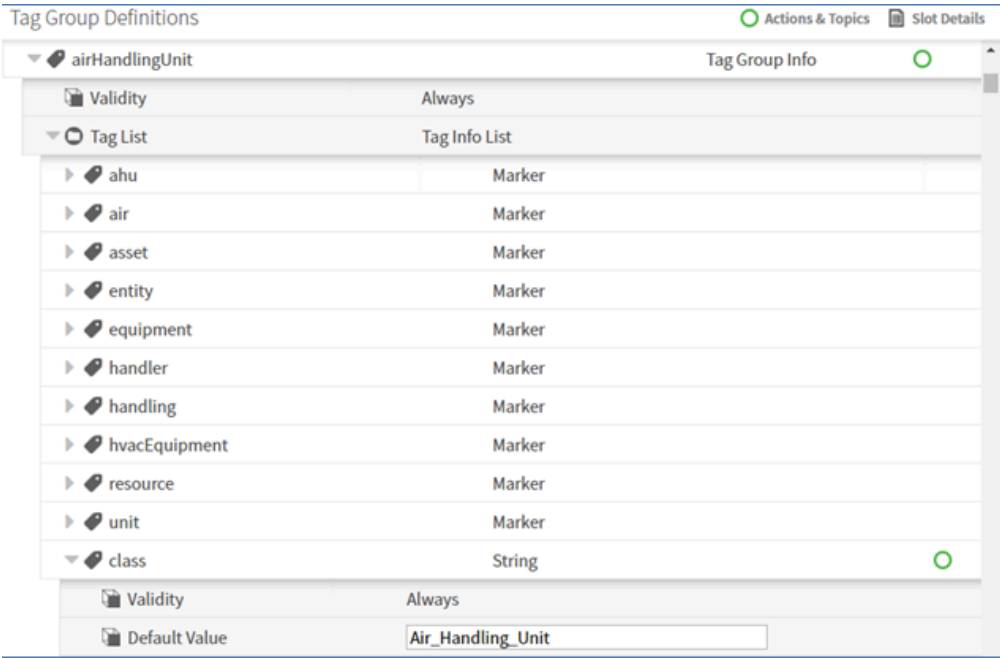
A Brick schema organizes entities of a building into a class hierarchy, where each level is a more specific version of its parent. In Brick, the types are defined as a hierarchy of classes.

The Brick tag dictionary contains a library tag groups, relations, and rules that enable a Niagara station to be modelled using Brick semantics defined in the Brick ontology. As a station is being constructed, these dictionary elements can be applied, or they can be applied to components of an existing station. The Brick tag dictionary does not contain any custom properties or actions, and it has no custom views associated.

Module information: The brick-rt module, which needs to be installed on your station, implements a Niagara Smart tag dictionary to support modelling a system using the Brick schema. This module has a dependency in the tagdictionary-rt module, which is required for any tag dictionary.

The Brick tag dictionary contains the following definitions for tag groups, relations, and tag rules:

- **Brick subclasses:** Brick schema entities can be declared a subclass of another entity. These are modeled as tag groups in the dictionary. The subclass hierarchy elements are included as tags in a tag group. For example, the **airHandlingUnit** tag group includes subclass entries for **hvacEquipment**, **equipment**, and **entity**.
- **Brick hasAssociatedTag:** Many Brick schema entities have a **hasAssociatedTag** property. These are modeled in the dictionary as tag groups. For example, the **airHandlingUnit** tag group includes tags for **air**, **equipment**, **handler**, **handling**, and **unit**.
- **Brick aliasOf:** A few Brick schema entities are declared as an alias of another entity. For example, the Brick AHU entity is an alias of the Air_Handling_Unit entity. Alias entities are included in the Brick dictionary as tag groups with the same contents as the originating entity tag group.
- **Brick class:** Each tag group includes a class tag with a String value of the Class name declared in the Brick schema. For example, the **airHandlingUnit** tag group contains a **class** tag with the value "Air_Handling_Unit".
- **Brick Id:** Starting in Niagara 4.15, the **id** tag has been removed from the dictionary.
- **Brick inverse relations:** Most of the relationships defined in the Brick schema have associated inverse relationships. For example, if one entity has a **bk:isPartOf** relationship to another entity, there is an associated inverse **bk:hasPart** relationship. In these cases, when you apply one of these relations between two entities in a station, the dictionary rules will imply the inverse relation between those two entities.



NOTE: The Haystack smart tag dictionary and the Brick tag dictionary can run simultaneously.

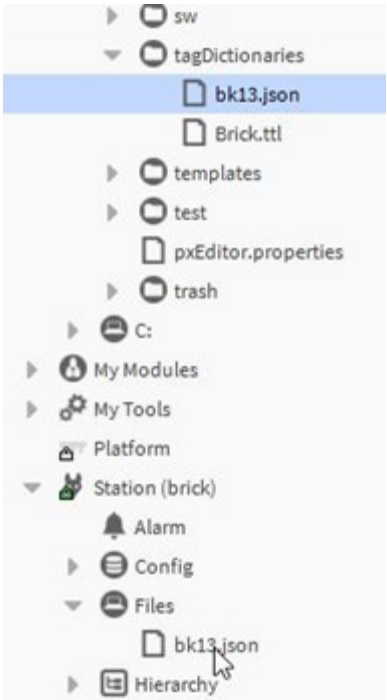
Updating Brick tag dictionary

You can update the Brick tag dictionary version outside of the official Niagara release schedule. When the Brick consortium releases a new version of the ontology, Tridium will generate a new version of the Brick tag dictionary as a JSON file and make it available for customers. The Niagara Brick tag dictionary supports the most recent version of the ontology with each Niagara release.

Prerequisites:

The Brick tag dictionary is installed on the station.

- Step 1. In the Workbench Nav tree panel, locate the JSON file and copy it to the station by dragging it to the station's **Files** container.



Step 2. To load the file into a Brick tag dictionary on a station, from the view selector open the **AX Slot Sheet** view on the Brick tag dictionary in the **TagDictionaryService**, right-click on the **importDictionary** action row and select **Config Flags**.

<input type="radio"/>	Property	9	tagDefinitions	Tag Definitions	Frozen		tagdictionary:TagInfoList
<input type="radio"/>	Property	10	tagGroupDefinitions	Tag Group Definitions	Frozen		tagdictionary:TagGroupInfoList
<input type="radio"/>	Property	11	relationDefinitions	Relation Definitions	Frozen		tagdictionary:RelationInfoList
<input type="radio"/>	Property	12	importDictionaryOrd	Import Dictionary Ord	Frozen	h	baja:Ord
<input checked="" type="radio"/>	Action	13	importDictionary	Import Dictionary	Frozen	h	
<input checked="" type="radio"/>	Action	14	exportDictionary	Export Dictionary	Frozen		
<input type="radio"/>	Property	15	tagRules	Tag Rules	Frozen		

Add Slot

Copy

Delete

Rename Slot

Config Flags

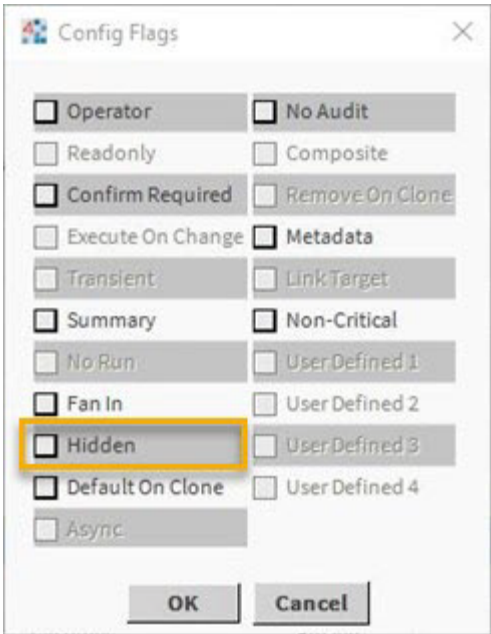
Ctrl+A

Ctrl+C

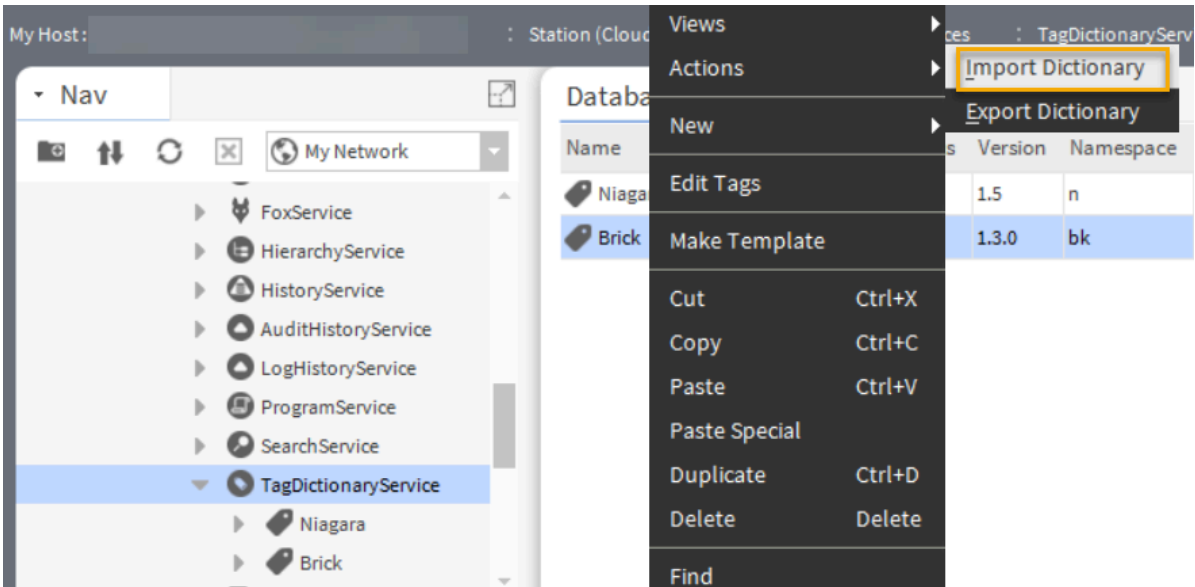
Delete

Ctrl+R

Step 3. Unselect the **Hidden** flag for the **importDictionary** action, and click **OK**.

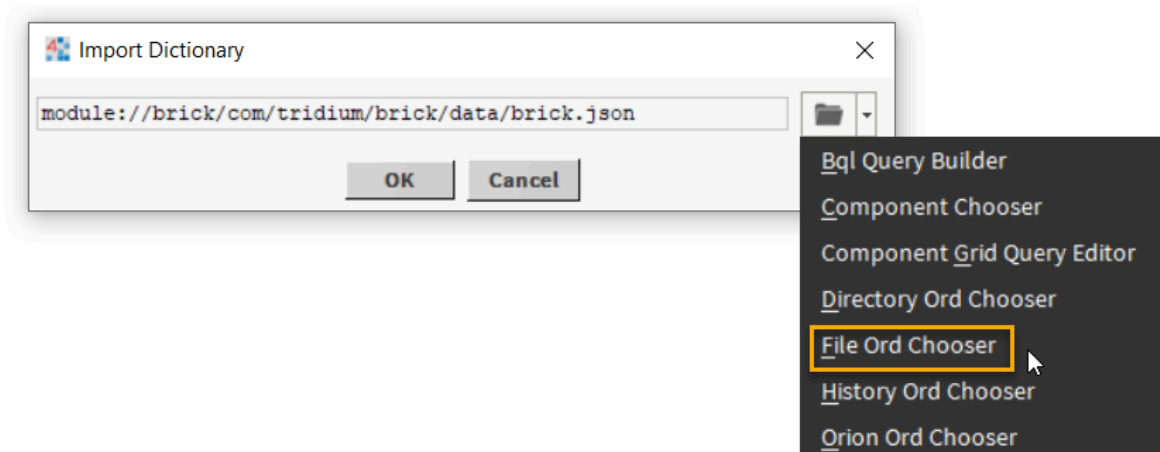


Step 4. Navigate back to the **Tag Dictionary Manager** view on the **TagDictionaryService**, and on the **Brick** tag dictionary, right-click and select **Actions > Import Dictionary**.

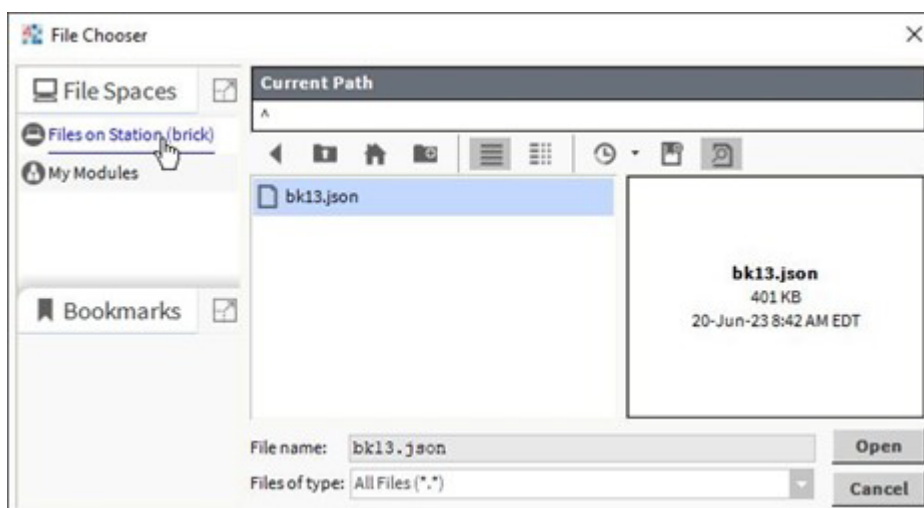


The **Import Dictionary** window opens.

Step 5. Click the down-arrow next to the folder icon and select **File Ord Chooser**.

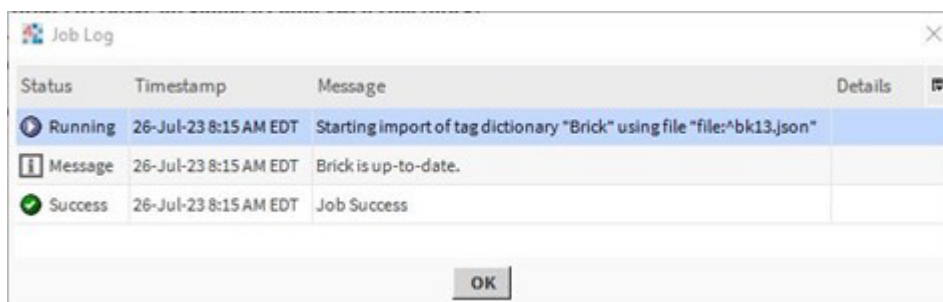


The File Chooser window opens.



- Step 6. From the **File Chooser > Files on Station**, select the JSON file already copied to the station and click **Open**.

The Brick tag dictionary import process runs as a job on the station. You can view the results of the import process by opening the **Job Service Manager** on the station's **Job Service** and clicking the double arrow icon on the right side of the **Tag Dictionary Import** row.

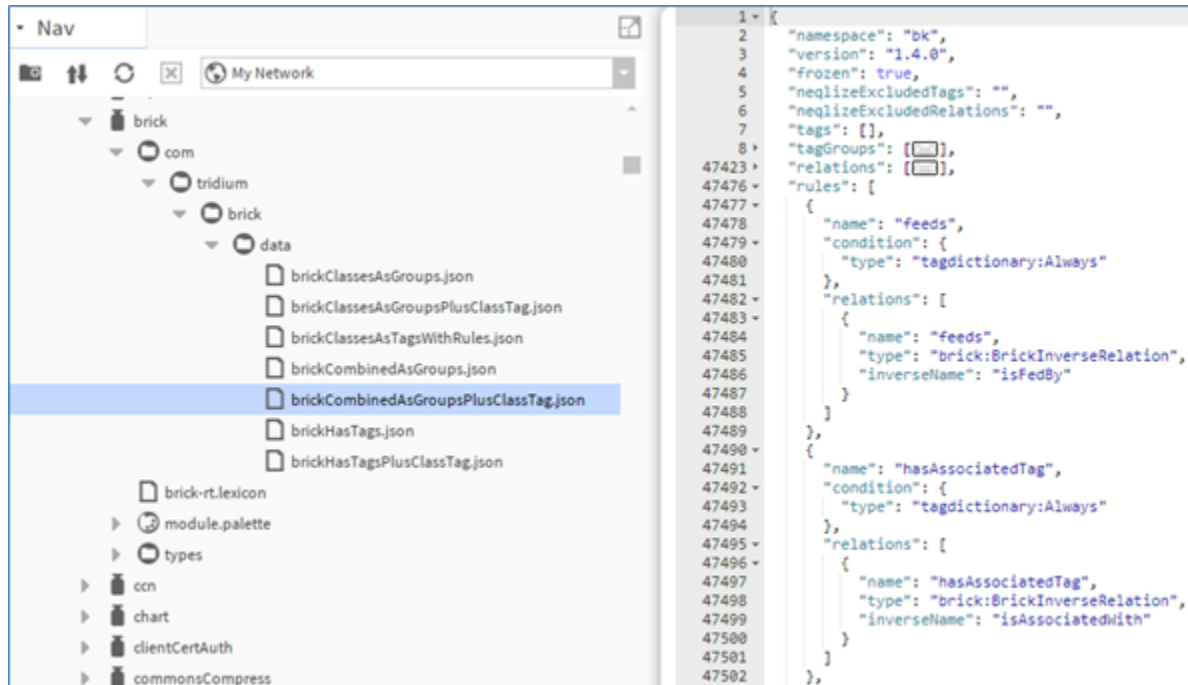


Brick custom rules

The JSON file loaded as part of the import process contains a list of rules to be included in the Brick tag dictionary. If desired, you can add to or override the default rules with a custom rule set.

To add or override the default rules with a custom rule set, you create their own JSON rules file and select it in the **Select a JSON file for custom rules (optional)** field in the **Generate Brick Dictionary** dialog window.

The brick module contains the default JSON file `brickCombinedAsGroupsPlusClassTag.json`. You can view it in Workbench by opening it in the Nav tree.



The existing Brick tag rules are defined in the rules object. The custom rules are defined as a JSON object with that same format. There is one JSON element for each Niagara property in a tag rule. Each tag rule has an entry for "name", "condition", and then at least one entry for "tags", "tagGroups", or "relations". These in turn have JSON elements for each of their respective Niagara properties.

NOTE: It is possible to declare a tag rule to be assigned to a specific Niagara type, which allows you to develop any custom rules necessary as JAVA classes.

An example of a custom rules file for three custom rules:

```
{
  "rules": [
    {
      "name": "id",
      "condition": {
        "type": "tagdictionary:IsTypeCondition",
        "objectType": "control:ControlPoint"
      },
      "tags": [
        {
          "name": "id",
          "type": "brickTest:BrickCustomIdTag",
          "validity": {
            "type": "tagdictionary:IsTypeCondition",
            "objectType": "control:ControlPoint"
          }
        }
      ]
    },
    {
      "name": "enableStatus",
      "condition": {
        "type": "tagdictionary:IsTypeCondition",
        "objectType": "control:BooleanPoint"
      },
      "tagGroups": [
        {
          "name": "enableStatus",
          "tags": [
            {
              "name": "enable"
            },
            {
              "name": "point"
            },
            {
              "name": "status"
            }
          ]
        }
      ]
    },
    {
      "name": "hasQUOTReference",
      "condition": {
        "type": "tagdictionary:Always"
      },
      "relations": [
        {
          "name": "hasQUOTReference",
          "type": "brickTest:BrickCustomQUOTRelation"
        }
      ]
    }
  ]
}
```



```
}
```

- The first id rule declares the condition as any ControlPoint, and the tag id will be applied. This tag has a custom JAVA implementation declared as a BrickCustomId type in a brickTest module. The JAVA class for this would be named BBrickCustomId.
- The second enableStatus rule has a Boolean Point condition and adds a tag group that contains the enable, point, and status tags.
- The third hasQUDTReference rule has an Always condition and adds a relation with a custom BrickCustomQUDTRelation type in a brickTest module.

Chapter 4. Tagging reference

Tagging is a form of semantic modeling that assigns information (one or more tags) to objects. The tag information can help integrators and users significantly when searching for objects, designing system structures or navigating hierarchies.

Tagging can identify a device and indicate where it is physically located. By identifying and locating devices, tags provide a context for the device that can be used in many different ways. When you use tags, you can reduce or eliminate the requirement to manually map objects directly to a desired application.

About tags

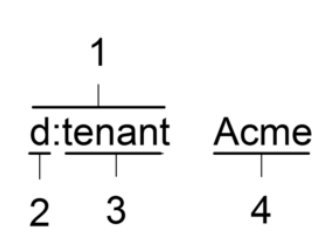
Tags assign additional information to objects in order to make the objects more accessible and flexible for search and system design. Tags also facilitate the design and use of hierarchical organization in a station user interface, whether you are working with an Enterprise Supervisor station or a single controller station.

NOTE: The tags available for use are defined in the tag dictionaries installed on your station.

Tag structure

A tag contains different parts that, together, make the tag useful as additional information on objects in a station. The following diagram shows the four basic parts of a tag.

Figure 16.Parts of a Tag



The following table provides definitions of the different parts of a tag:

Item	Tag Element	Description
1	Tag Id	The tag Id is comprised of a dictionary and name, generally displayed as two pieces of text separated by a colon: dictionaryNamespace:name.
2	Tag dictionary	The dictionary string is used to link or assign a tag to a particular "namespace" (tag dictionary). This is typically a very short string of only a few characters. NOTE: If the dictionary is not defined (empty string), the Id is displayed with just the name.
3	Tag name	The name string provides the semantic information and is often paired with the tag value.
4	Tag value	A string value assigned to the tag for more information, for example: building name, device name, location, or other.

Types of tags

The following table describes types of tags that may be used on the system:

Tag type

Description

Direct tags

Direct tags are tags that you add intentionally to a component using an installed tag dictionary or an Ad Hoc tag. In its simplest form, a tag on a component is a component “property”, with a non-component value and a metaData flag set. The property name is a string form of the tag Id. In the **Edit Tags** dialog box, direct Ttags are listed under the **Direct Tags** tab.

Implied tags

Implied tags are tags that are not directly stored in the component, but are implied by tag rules that are defined in installed Smart Tag Dictionaries. These tags are typically the remapping of existing component properties to the semantic naming convention defined in a tag dictionary. In the **Edit Tags** dialog box, Implied tags are listed under the **Implied Tags** tab.

Ad Hoc tags

An Ad Hoc tag, also a direct tag, is one that you create in the **Add Tag** dialog box just before adding it to a component. Ad hoc tags are not included in any tag dictionary.

Online tagging versus offline tagging

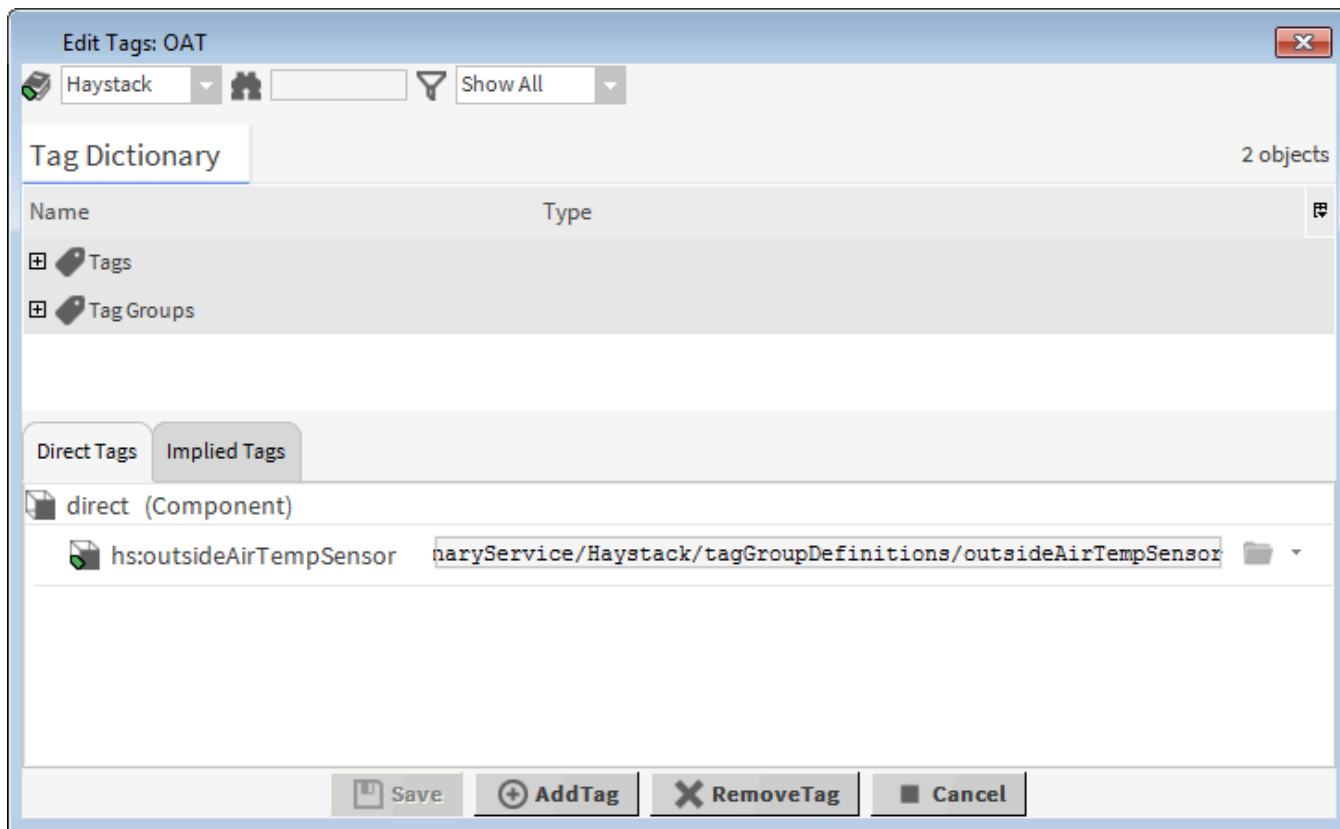
There are three separate scenarios in which you apply tags:


- **Online tagging:** The installed tag dictionaries in `TagDictionaryService` take effect.
- **Offline tagging in a station with tag dictionaries:** The installed tag dictionaries in the `TagDictionaryService` take effect. You will also see implied tags in the **Edit** dialog.
- **Offline tagging in a station when no dictionaries are found:** The system searches for tag dictionaries in the following locations:
 - all palettes of installed tag dictionary modules
 - in the `user-home/tagDictionary` folder where custom tag dictionaries are stored
 - also searches for implied tags

About the Edit Tags dialog

The **Edit Tags** dialog as well as the station and point manager views are the primary methods for adding a tag or a tag group to a component. The dialog lets you add individual tags or tag groups to the object, as well as remove them from the object. The lower half of the dialog provides tabs to view the direct and implied tags assigned to the object.

Invoke the **Edit Tags** dialog box by right-clicking an object and selecting **Edit Tags**.


Figure 17. Edit Tags dialog

In the **Edit Tags** dialog box, select a dictionary from the option list in the top left corner .

TIP: You can use this shortcut to select a dictionary. In the **Search** field type `hs:` for Haystack, `n:` for Niagara, or enter the namespace for another dictionary.

The top half of the dialog box shows a list of tags available from the selected dictionary. Once a tag is assigned to the active object, the tag icon appears dimmed.

To facilitate making selections, the dialog box includes filters that help by narrowing the list of tags from which you can choose. This is most useful when selecting from a dictionary containing a huge number of tags, such as the Haystack Tag Dictionary.

Type in the **Search** field  to filter by tag name. Tags are filtered immediately as you type.

- If the list has only a single item, it is selected by default.
- If a tag name is a subset of another tag, adding a space selects the shorter tag by name.
For example, if you have both "chiller" and "chillerPlant" tags, typing "chiller" shows both tags. Adding a space after "chiller" filters out "chillerPlant" and shows the "chiller" tag only.
- Entering a colon ":" filters for the tag dictionary that has the prefixed namespace.
For example, if you enter "hs:temp" you select the "hs" (Haystack) dictionary and the "temp" tag.



You can also select an option from the option list to filter based on validity options.

- **Show All:** no filtering applied when this option is selected.
- **Valid Only:** shows just the tags that are valid based on rules defined in the tag dictionary.
- **Best Only:** filters tags in appropriate manager views based on the identity of the component; for example, whether it is a point or device.

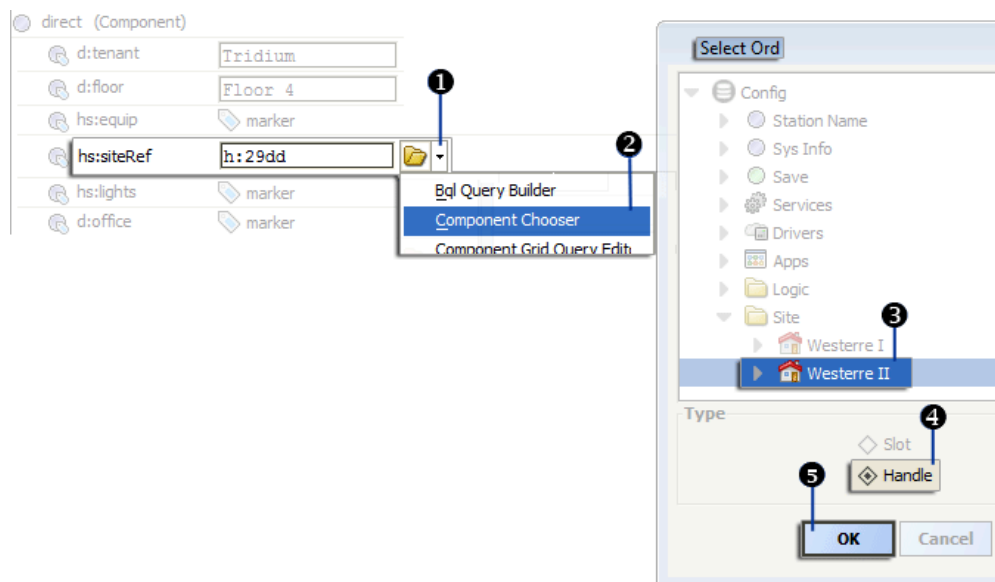
Methods for adding tags include the following:

- Add an individual tag from a dictionary.
- Add a tag group (predefined collection of tags) from a dictionary.
- Add a unique ad hoc tag which you create.

After clicking **Add Tags**, the selected tags are added and appear in the **Direct Tags** table in lower half of the dialog box.

After editing any tag value fields as needed, click **Save** to save the tag assignments.

For tags that have Ord type values such as "hs:siteRef", refer to the following image and steps as an example of how to add a link to your tag.



1. Click the option list arrow located to the right of the tag value field.
2. Select the appropriate link type from the options menu.
3. Browse to the desired link and select it.
4. Select the **Handle** option.
5. Click **OK**.

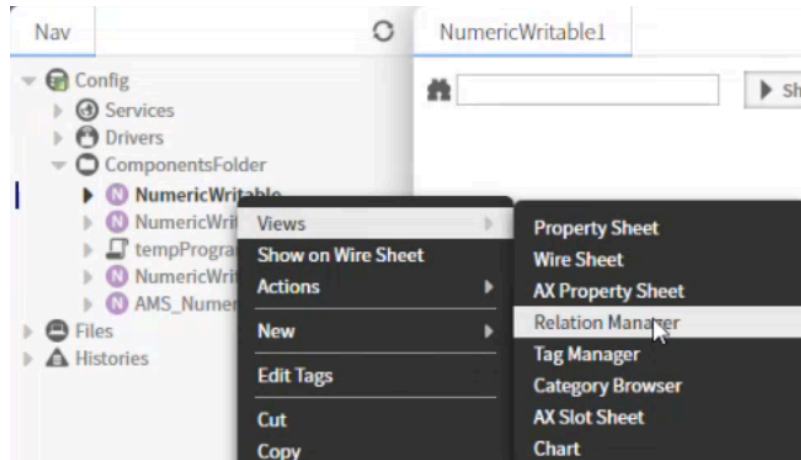
Relation Manager

The Relation Manager is an HTML5 view that you can use to add, edit, and delete relations and links (as of Niagara 4.15).

It is available as a view on all components. You can select it using the View Selector in the top-right corner of the screen.

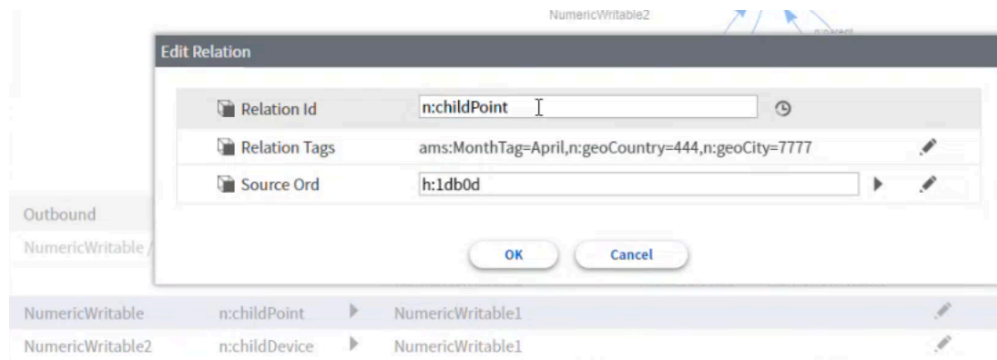


To view it, you can also right-click on a component and select **Views > Relation Manager**.

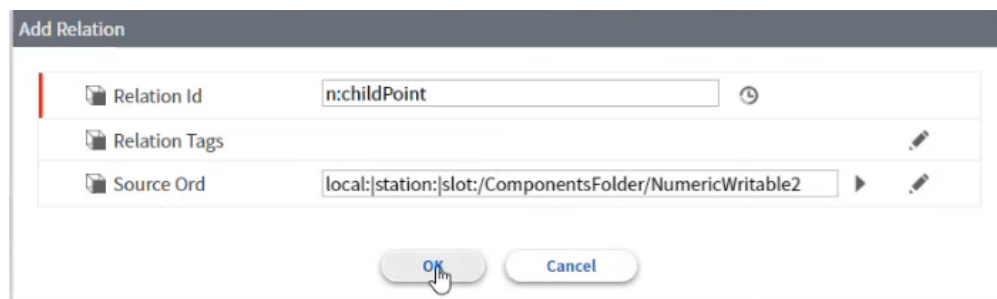


It is equivalent to the Relation Sheet in Workbench and offers the following features:

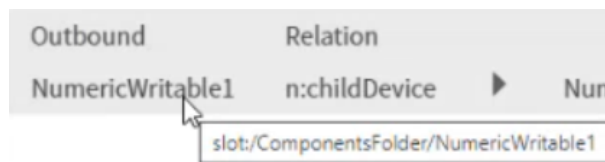
- Display implied and direct relations
- Show knobs and relation knobs
- Apply filters
- Visualize the relations
- Navigation using the visualization
- Edit relations and links



- Drag and drop a component from the Nav tree to the relation table to add a component. The dialog box that opens when you drop the component auto-populates with the Ord that you have dragged.



- Show slot location by hovering over it



Related documentation, see "Relation Sheet view" in Niagara Relations Guide.

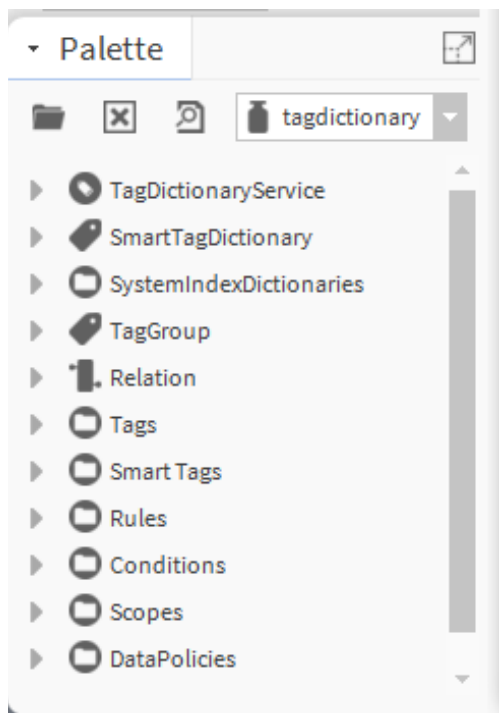
Components and views in the tagdictionary module

Components include services, folders, and other model building blocks associated with a module. You drag them to a property or wire sheet from a palette. Views are plugins that can be accessed by double-clicking a component in the Nav tree or right-clicking a component and selecting its view from the **Views** menu.

The component topics that follow appear as context sensitive help topics when accessed by:

- Clicking **Help > On View** (F1) while the view is open.
- Clicking **Help > Guide On Target**

Components in the tagdictionary palette are described in the following sections.

Figure 18. The tagdictionary palette

tagdictionary-TagDictionaryService

The **Tag Dictionary Service**, located in a station's Services directory, is the container for all tag dictionaries installed in the station.

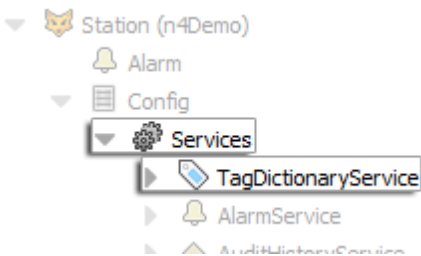
Tag Dictionary Service

The **Tag Dictionary Manager** is the main view for the Tag Dictionary Service. The service has a property for defining a **default namespace** so that queries that do not specify a namespace are resolved on this default namespace. For example, if you execute a NEQL query for "point" (instead of "n:point"), and the default namespace ID is set to "n", your query returns all objects tagged with "n:point".

Note the following information about the Tag Dictionary Service:

- The **tags** license is required to use the TagDictionaryService and tag dictionaries on a station.
- A station can support only one tag dictionary service.
- Neither the tag dictionary service nor any tag dictionaries are strictly required for tagging or for performing NEQL queries. However, without tags or tag dictionaries, a station is not be able to take advantage of most of the functions available from tagging objects. Tags and tag dictionaries are fairly lightweight and therefore are included by default in all stations created by the new station wizard.
- Installed tag dictionaries belong under the **TagDictionaryService**. They are not allowed anywhere else in the station.

Figure 19. TagDictionaryService is located in the Services directory



Tagging enhancements include added support for tag-based Px bindings which resolve NEQL query Ords instead of slot path Ords. The purpose of this is to enable you to create reusable graphics that can be installed on any component in any station where the bound components are properly tagged and related correctly to the base component. Tag-based Px bindings use a NEQL query and a new single scheme that resolves the query result to a single entity. The TagDictionaryService includes new configuration properties to support this functionality. For details, see the “Neqlize options” section.

NOTE: The tagdictionary palette includes two smart tag dictionaries that support the System Database and System Indexing features. The dictionaries allow you to easily exclude portions of a NiagaraStation from indexing operations, which you might want to do for licensing reasons. This dictionary has a smart tag, the scoped tag. If you add this dictionary to your station, you can drop a `systemIndex:excluded` marker tag on a component and any descendants will have this same tag implied on them. To prevent implying this tag on all descendants, you can apply a `systemIndex:included` marker tag where necessary. This dictionary has a scoped tag rule that can be configured to imply the `systemIndex:excluded` tag on portions of the station as specified in one or more Ord scopes.

The Tag Dictionary Service includes the following indexing features both of which reduce the amount of re-evaluation of tag rules, which improves response time for obtaining results of NEQL searches and traversing hierarchies:

- Tag Rule Index, enabled by default. For complete details, see [Tag Rule Index](#).
- The Implied Tags Index, disabled by default, can be configured for user-selected implied tags. For complete details, see [Implied tags index](#).

Tag Dictionary Service properties

Property	Values	Description
Enabled	true or false	Activates and deactivates use of the function.
Status	Text, read-only	Indicates the condition of the component at last polling. <ul style="list-style-type: none">• {ok} indicates that the component is polling successfully.• {down} indicates that polling is unsuccessful, perhaps because of an incorrect property.• {disabled} indicates that the <code>Enable</code> property is set to false.

Property	Values	Description
		<ul style="list-style-type: none"> • fault indicates another problem.
Fault Cause	Text, read-only	Indicates why the network, component, or extension is in fault.
Default Namespace Id	Text, string	This TagDictionaryService property provides a field that is used to indicate a default tag namespace (tag dictionary), which is used if there is no namespace provided as part of a query. For example, if the default namespace is set to "hs", a search query that includes only "ahu" would return all objects that are tagged with "hs:ahu".
Tag Rule Index Enabled	true (default) or false	Enabled by default, the Tag Rule Index is an index on the Tag Dictionary Service which improves performance in evaluating tag rules for implied tags during NEQL searches. Setting the property to false automatically clears the index, as do any changes in the service.
Indexed Tags	text string	Entering tag ids in this field enables those tags to be indexed for each component in the station. Use a semi-colon separated list for multiple tag ids that should be indexed. This field can be cleared to clear the implied tag index and prevent any further indexing.
Neqlize Options	additional details	Contains sub-properties that list default excluded relations and tags, enable/disable use of the default exclusions, custom excluded relations and tags. You can specify whether to use the default exclusions along with any custom exclusions or to use only the your custom exclusions. For details, see the following section on "Neqlize Options".

Actions

- **Clear Tag Rule Index** allows you to manually clear the Tag Rule index.

NOTE: Typically, you would not need to invoke this **Clear Tag Rule Index** action because the index is cleared automatically whenever changes are made in the Tag Dictionary Service. It is provided so that you can reset everything when you are not seeing the expected results.

- **Invalidate All Tag Indexes** resets (clears) all tag indexes.
- **Invalidate Single Tag Index** resets (clears) the specified tag indexes.

NOTE: These actions do not discontinue indexing for the tagIDs listed in the **Indexed Tags** property. The index will be rebuilt the next time a search for one of the indexed tagIDs is executed.

Neqlize options

There is added support for tag-based NEQL query Ords. The TagDictionaryService features several added properties to specify certain tags and relations to be excluded when converting slot path Ords to NEQL query Ords.

Figure 20. Neqlize Options properties on the TagDictionaryService

Neqlize Options

Neqlize Options

Default Excluded Relations

n:child, n:parent, n:tagGroup

Use Default Excluded Relations

true (Append Custom to Default)

Custom Excluded Relations

Default Excluded Tags

n:bindHints, n:displayName,
n:history, n:name, n:node,
n:ordInSession, n:station,
n:targetSlotHint, n:type, n:vendor,
n:version

Use Default Excluded Tags

true (Append Custom to Default)

Custom Excluded Tags

Refresh

Save

For example, a tag-based NEQL query Px Ord binding using the **n:name** tag would hurt the reusability of a graphic because the bound component would have to be named the same under a different base component. Using the **n:ordInSession** or Haystack **hs:id** tag would be equivalent to using an absolute slot path Ord. Using the parent or child implied relations would be somewhat more limiting than relative slot path Ord.

The **Default Excluded Relations** and **Default Excluded Tags** values are collected from each installed tag

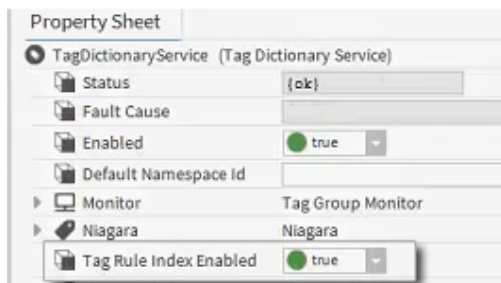
dictionary. **Custom Excluded Relations** and **Custom Excluded Tags** values can be optionally specified and be either appended to or replace the default values. User values for **Excluded Relations** and **Excluded Tags** can be specified in the Px Editor **Workbench Options** and either appended to or replace these TagDictionaryService values.

Name	Value	Description
Default Excluded Relations	string	Default values collected from installed tag dictionaries for relation pattern filters used to exclude relations when converting slot path Ords to traverse NEQL query Ords.
Use Default Excluded Relations	true (default), false	When <code>true</code> , the custom values for relation pattern filters will be appended to the default values. Otherwise, the custom values will be used exclusively and the default values will be ignored.
Custom Excluded Relations	string	Custom values for relation pattern filters used to exclude relations when converting slot path Ords to traverse NEQL query Ords.
Default Excluded Tags	string	Default values collected from installed tag dictionaries for tag pattern filters used to exclude tags when converting slot path Ords to NEQL query Ords.
Use Default Excluded Tags	true (default), false	When <code>true</code> , the custom values for tag pattern filters will be appended to the default values. Otherwise, the custom values will be used exclusively and the default values will be ignored.
Custom Excluded Tags	string	Custom values for tag pattern filters used to exclude tags when converting slot path Ords to NEQL query Ords.

Tag Rule Index

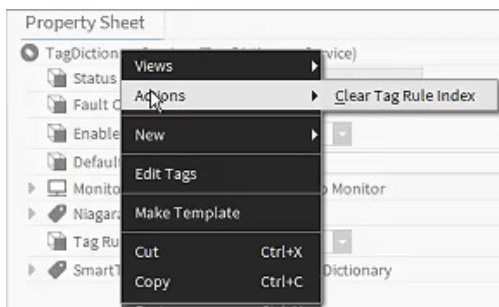
The tag rule index, which is enabled by default, is an index on the **Tag Dictionary Service**. The index improves performance in evaluating tag rules for implied tags during NEQL searches. The index, which is built as NEQL queries are executed, maps tags to the tag rules that imply those tags. This index should not require a significant amount of system memory.

NOTE: The type of memory being used by the tag rule and implied tag indexes is heap memory. So these indexes are not stored persistently but are built dynamically after every station reboot when NEQL queries are submitted.

Figure 21. Tag Rule Index Enabled property

To ensure that the index is kept up-to-date, it is cleared automatically whenever changes are made in the **Tag Dictionary Service**. For example, if you delete a tag from the tag list under a tag rule, the index is automatically cleared. This prevents incorrect or incomplete results from occurring in your hierarchies, searches, and anything else that uses NEQL queries. As you execute subsequent searches the index is rebuilt.

Although cleared automatically, there is an added **Clear Tag Rule Index** action on the **Tag Dictionary Service** which allows you to manually clear the index. Typically, you would not need to invoke this action. It is provided so that you can reset everything when you are not seeing the expected results. It is not likely that the tag rule index would be the cause of the problem but the action is available just in case.

Figure 22. Clear Tag Rule Index action

Additionally, within the **Spy Remote** view, there is an added "Tag rule index info" section, which lists details of the index. Specifically, it shows the following:

- the number of implied tags, technically, tagIDs in the index referred to as "# of indexed tags"
- the number of tag rules in the index implying those tags; a single rule may appear many times in the index and indicates via these values whether the index has been cleared. For example, when the index is cleared or disabled both the "# of indexed tags" and "# of tag rules" values on the Spy page will be zero.

Figure 23. Tag rule index info section in Spy Remote view

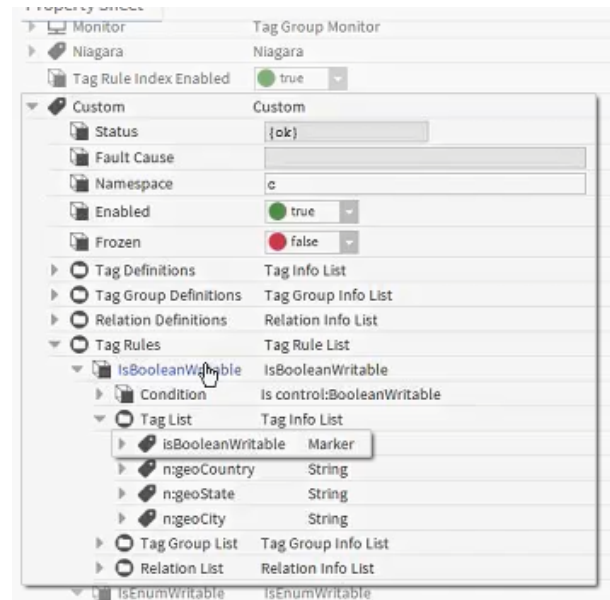


Imply tags in a different tag dictionary

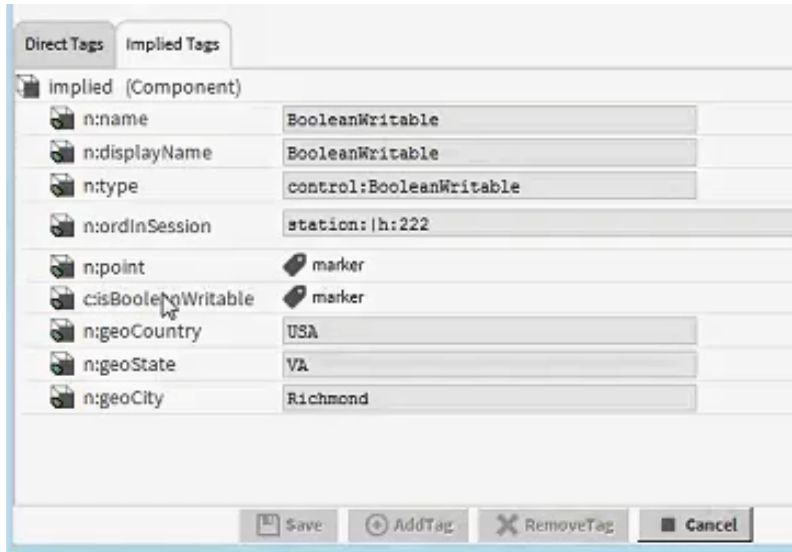
Tag rule index allows you to imply tags in a different tag dictionary than the one containing the tag rule doing the implying. Stated another way, this permits you to use tags from other tag dictionaries with a different namespace, such as the Niagara tag dictionary which is frozen in the tag rules of your custom smart tag dictionaries. Frozen means that you cannot add rules that persist through a station restart.

For example, the following image shows the property sheet for a custom smart tag dictionary in which a tag rule contains several tags, one of which is a marker tag for “isBooleanWritable”. Notice that this tag is not fully qualified, meaning that the tag name does not include the namespace of the source tag dictionary. In this situation the tag automatically assumes the namespace of the dictionary in which it is being used, in this case the “Custom” tag dictionary: c:isBooleanWritable. The other tags in the tag rule are fully qualified and reference a completely different tag dictionary, the Niagara tag dictionary as indicated by the namespace, n:.

Figure 24. Tag dictionary property sheet followed by implied tags in Edit Tags dialog



Right-clicking a booleanWritable point in the station allows you to select the Edit Tags dialog box. On the Implied Tags tab, you can see that the tags in the example tag rule are implied on this object. Also, you can confirm that the tag rule has implied tags from the parent smart tag dictionary (Custom, c:isBooleanWritable) as well as tags from a different tag dictionary (Niagara, n:geo*).

Figure 25. Implied Tags tab in Edit Tags dialog

More significantly, when you search for "n:geoCountry", for example, you will get booleanWritable points in the results.

Implied tags index

Implied tags index improves performance of NEQL searches and hierarchy traversal. Implied tags originate from evaluating the tag rules in smart tag dictionaries and evaluating direct tag groups on an entity. Indexing implied tags limits the amount of tag rule re-evaluation that occurs on subsequent NEQL queries.

For example, a NEQL search on an unindexed tag (ex.: c:city) requires that every tag rule that contains this tag be evaluated; every component throughout the station be evaluated for this tag; and this evaluation must be done every time a search for this tag is initiated. All of which adds up to a significant drain on performance.

JACE is constrained not only by processor power but also by memory availability. To control the impact on memory, the Implied Tags Index is disabled by default.

NOTE: The type of memory being used by the tag rule and implied tag indexes is heap memory. So these indexes are not stored persistently but are built dynamically after every station reboot when NEQL queries are submitted.

You can enable specific tags for indexing by entering those individual tagIDs in the **Indexed Tags** field in the **Tag Dictionary Service** property sheet. Entering tagIDs in this field enables those tags to be indexed for each component in the station. Use a semi-colon separated list to enter multiple tagIDs to be indexed. Tags that are not enabled for indexing will use normal tag rule and direct tag group evaluation.

To stop indexing from occurring on some or all tagIDs and reduce the memory used by this index, remove tagIDs from the **Indexed Tags** field in the **Tag Dictionary Service** property sheet.

Although there is no mechanism for clearing the index, you can reset the index whether a tag definition is implied or not via the following actions on the **Tag Dictionary Service**. These actions do not discontinue indexing for the tagIDs listed in the **Indexed Tags** field but reset the index in regards to some or all of the tagIDs being indexed. The index will be rebuilt the next time a search for one of the indexed tagIDs is executed.

- Invalidate All Implied Tags Indexes
- Invalidate Single Implied Tags Index

CAUTION: Memory use is strongly influenced by the number of components in the station and then also influenced by the number of tags being indexed. The implied tags index has the potential to grow large enough to exceed available memory. To avoid this, limit the number of tags enabled for indexing.

Either direct or implied tags can be added to the **Indexed Tags** field. If an object contains an indexed tag as a direct tag, the index will not be used at all. If the entity does not contain the indexed tag as a direct tag, a search through the rules will be executed to determine if the tag is implied and the index will store whether or not the tag is implied.

NOTE: The implied tag index is most helpful for tags that are implied (rather than direct tags), because the re-evaluation of any tag rules can be skipped.

Implied tags that are safe for indexing are those tags that will be implied and whose values will not change regardless of changes to the station. For example, tags that are related to the type of a component, which can never change, are very safe to index. Tags related to type of a component include:

- n:type
- n:point
- n:input
- n:output
- n:device
- n:network
- n:schedule
- hs:connection
- hs:cur
- hs:device
- hs:equip
- hs:kind
- hs:network
- hs:point
- hs:writable

Tags used in GroupLevelDefinitions are good candidates for indexing because they are used frequently. However, that does not necessarily make them safe tags to index.

Other implied tags may be riskier because rule conditions and contents can be changed and the changes might not be reflected in the index. Tags that are implied using the HasAncestorRule or HasRelationRule conditions depend on the station configuration and, if the station configuration changes, might affect whether or not tags are implied.

Also, it is not the value of the tag that is indexed but the tag definition. So, if the value of the tag depends on something that changes, those changes will be reflected in the value of the tag when the tag is retrieved from the index. For example, if the history ID of a history extension changes, the new value will be reflected in the value of the n:history tag when it is retrieved.

Tags whose values are derived based on component and/or properties of the station are called smart tags. Other smart tags that are safe to index include the following:

- n:name
- n:displayName
- n:ordInSession
- n:station
- hs:curErr
- hs:curStatus

- hs:curVal
- hs:enum
- hs:his
- hs:hisErr
- hs:hisInterpolate
- hs:hisStatus
- hs:id
- hs:maxVal
- hs:minVal
- hs:tz
- hs:unit
- hs:writeErr
- hs:writeLevel
- hs:writeStatus
- hs:writeVal

NOTE: If you find that the JACE is at the limit of memory usage, it is best to clear any tags from the **Indexed Tags** field in the **Tag Dictionary Service** so that the indexing no longer occurs. If the station crashes before the **Indexed Tags** field can be cleared, the station can be started with the system property `niagara.tagdictionary.disableTagIndexing` set to true. This will prevent any indexing and allow the **Indexed Tags** field to be cleared. Then, this system property can be cleared and the station restarted to allow indexing again.

NiagaraTagDictionary

The NiagaraTagDictionary is a frozen slot of the Tag Dictionary Service.

This property is a smart tag dictionary containing a collection of tags developed for Niagara systems that are used for semantic modeling of specific building control entities, that is, networks, devices, equipment, points, sites, buildings, geo-location, histories. Since this is a smart tag dictionary, it applies implied tags and implied relations to components and links throughout the station. This allows queries to find these components based on type, linkage, hierarchy or combinations of these. The Niagara Tag Dictionary is indicated by the `<n>` namespace (the letter `<n>` followed by the colon character).

About tag dictionaries

The tagdictionary module contains tag dictionary components which you can use to create custom tag dictionaries. A tag dictionary is the container for a collection of tag definitions, tag group definitions, and relation definitions. The tags in a tag dictionary may be associated with devices, components, and points. Typically, these associations are established when the device is discovered, registered, and fully subscribed but tags can be added to an object at any time. Tags also provide a vocabulary for searching.

NOTE: The tags license is required in order to use the TagDictionaryService and tag dictionaries on a station.

The **Tag Dictionary Manager**, the primary view of the Tag Dictionary Service, displays the dictionaries that are installed on the station. You can create and add custom tag dictionaries to the station via this view (or by dragging from the tagdictionary palette). For example, you may create one or more custom dictionaries for a specific customer, for an OEM, or for a specific application.

Tag dictionary composition

A tag dictionary is composed of the following:

- A unique namespace, normally 1- or 2-characters, for example "n" for Niagara, "hs" for Haystack
- Tag definitions (contain individual Tag components added to the dictionary). This is the collection of

standardized tags with an Id.name that has semantic meaning for the given domain or namespace. The dictionary also defines tag default values and any validation rules for applying tags.

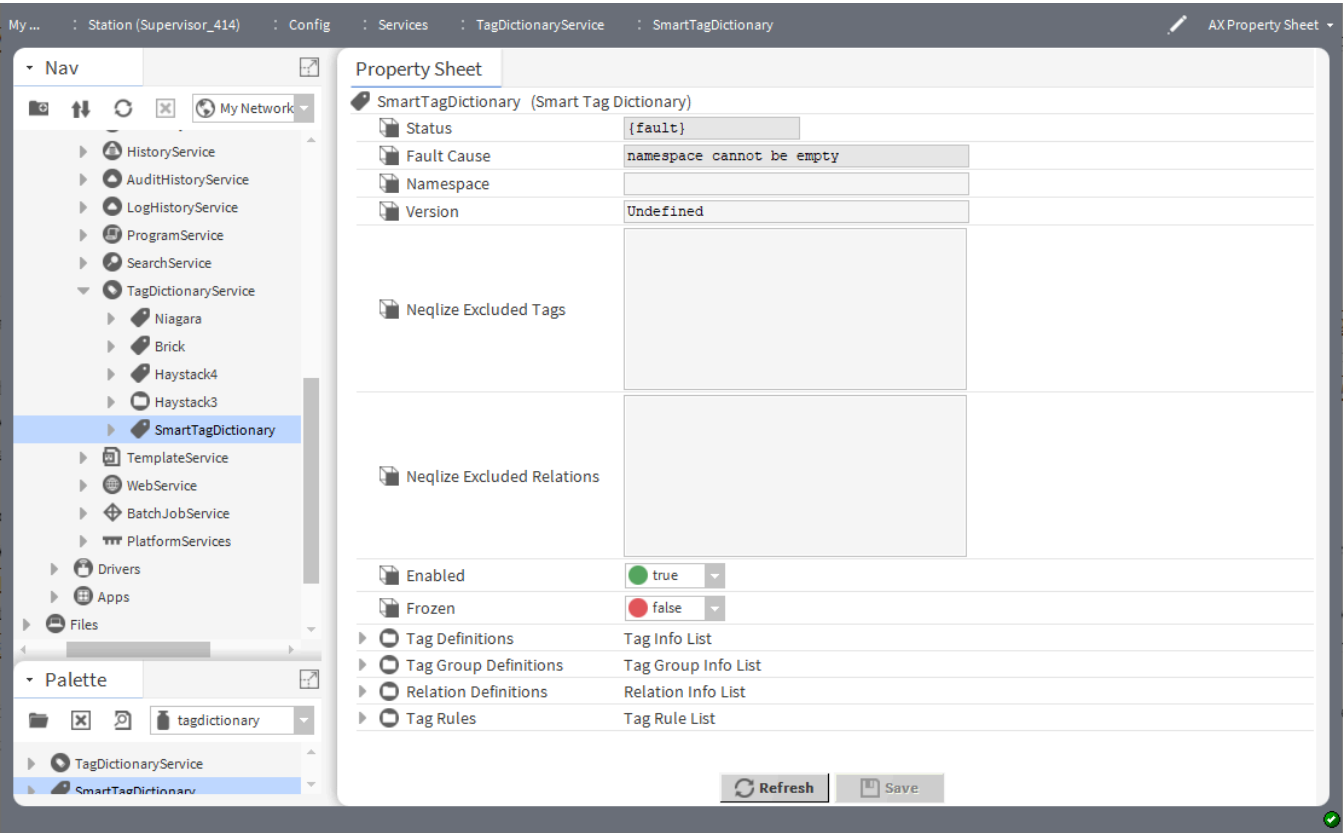
- TagGroup definitions (optional and contain individual TagGroups components added to the dictionary). This is a collection of standardized groupings of tags (tag groups) that have semantic meaning for the given domain or namespace. The dictionary also defines any validation rules for applying these tag groups.
- Relation efinitions (optional and contain individual relation components added to the dictionary). This is a collection of standardized relation Ids with semantic meaning for the given domain or namespace.
- TagRules (optional and in a smart tag dictionary only; they contain individual TagRule components added to the dictionary)

tagdictionary-SmartTagDictionary

The Smart tag dictionary automatically applies the implied tags and relations to objects. Implied items, that is, the implied tags and implied relations, are not added to the station, and the station size is not increased as a consequence.

Smart tag dictionary properties

To create a new Smart tag dictionary, drag the SmartTagDictionary component to the Tag Dictionary Service.



Property	Values	Description
Namespace	text	This field indicates the default tag namespace "bk" (tag dictionary).
Version	read-only	Displays the Smart tag dictionary version.

Property	Values	Description
Neqlize Excluded Tags	text string	Displays default values collected from installed tag dictionaries for tag pattern filters used to exclude tags when converting slot pathOrds to NEQL query Ords.
Neqlize Excluded Relations	additional details	Displays default values collected from installed tag dictionaries for relation pattern filters used to exclude relations when converting slot path Ords to traverse NEQL query Ords.
Enabled	true (default) or false	Enabled by default, the Tag Rule Index is an index on the Tag Dictionary Service which improves performance in evaluating tag rules for implied tags during NEQL searches. Setting the property to false automatically clears the index, as do any changes in the service.
Frozen	true or false (default)	If true, you cannot add rules that persist through a station restart.
Tag Definitions	additional details, tag info list	See <i>"Tag Definitions (TagInfoList)"</i>
Tag Group Definitions	additional details, tag group info list	See <i>"Tag Group Definitions (TagGroupInfoList)"</i>
Relation Definitions	additional details, relation info list	See <i>"Relation Definition (RelationInfo)"</i>
Tag Rules	additional details, tag rule list	See <i>"Tagdictionary-TagRuleList"</i>

Actions

Export Dictionary: Exports the contents of a tag dictionary to a CSV or JSON file to view and edit externally.

tagdictionary-SystemIndex

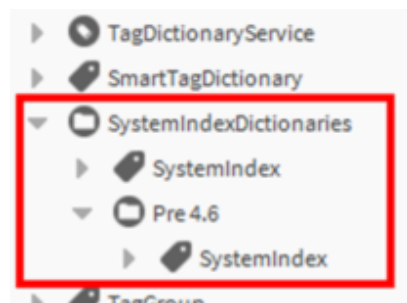
Added smart tag dictionaries in the tagdictionary palette provide support for the system database and system indexing features. Located in the SystemIndexDictionaries folder, the SystemIndex dictionary has a scoped tag to assist with excluding certain objects from system indexing operations.

Once you add the SystemIndex dictionary to your station, you can drop a `systemIndex:excluded` marker tag on a component and this tag will be implied on all of the component's descendants. In cases where you want to prevent implying this tag on some of the component's descendants, you can drop a `systemIndex:included` marker tag where necessary and those portions of the tree will not have the `systemIndex:excluded` marker tag implied and will not be excluded from system indexing.

For stations running prior versions

For stations running on older versions of Niagara, use the SystemIndex dictionary located in the `Pre 4.6` folder (shown).

Figure 26. SystemIndex smart tag dictionaries



This pre 4.6 SystemIndex dictionary has a scoped tag rule that can be configured to imply the systemIndex:excluded tag on portions of the station as specified in one or more ord scopes. The pre-4.6 SystemIndex dictionary only implies the systemIndex:excluded tag through ord scopes. The systemIndex:excluded marker tags will not be implied automatically when using the pre-4.6 SystemIndex dictionary.

Name	Value	Description
systemIndex:excluded	Marker	When applied to a component and when the SystemIndex dictionary for Niagara is enabled and installed, all descendants will have the same marker tag implied on them (except for descendants of a component with the systemIndex:included marker tag applied). This results in these components being excluded from system indexing.
systemIndex:included	Marker	When applied to a component, it prevents the systemIndex:excluded marker tag from being implied on that component or any of its descendants even if an ancestor of the component has the systemIndex:excluded marker tag. This results in the component and its descendants being included in system indexing.

Tag Definitions (TagInfoList)

The **Tag Definitions** folder in a tagdictionary contains the collection of standardized tags that have semantic meaning for that namespace (tag dictionary). Each tag in this **TagInfoList** can be used to add specific metadata to objects in a station, assigning additional semantic information, which provides a basis for searching. Tags typically contain a **Validity** slot with conditions such as, Always, IsType.

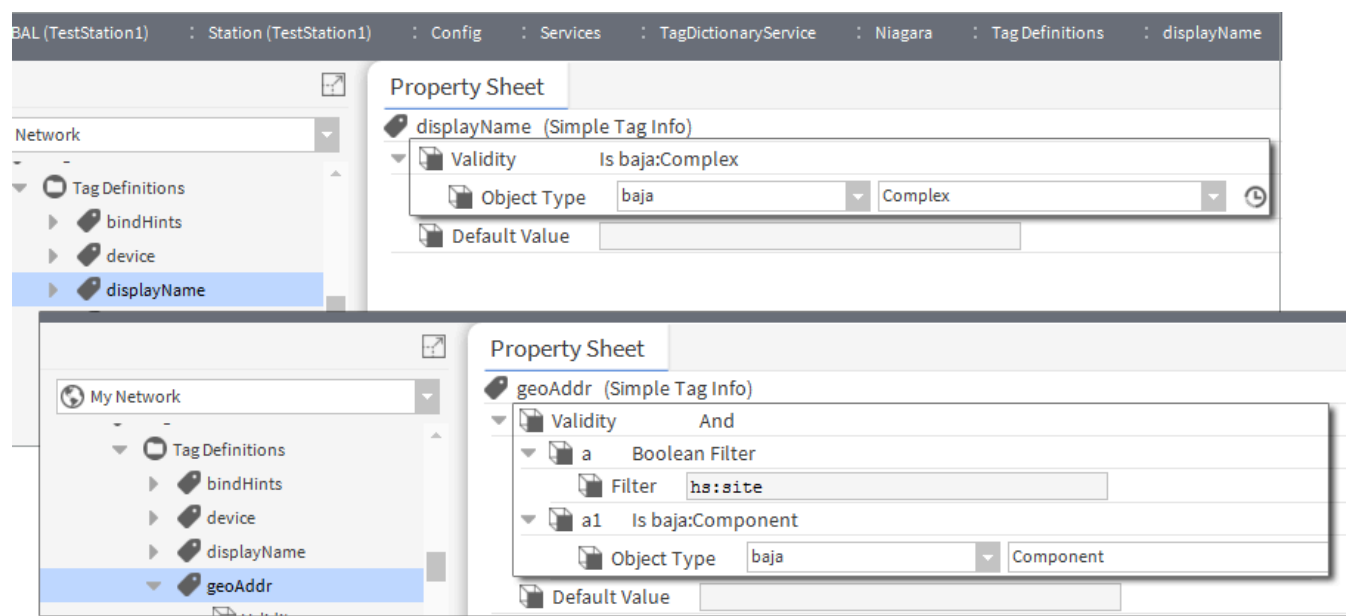
Tag properties

The following implementations of Simple Tag Info are available in the Tags folder of the tagdictionary palette. When creating a custom tag dictionary, drag and drop tags from the palette to the Tag Definitions folder in the tag dictionary’s property sheet to create the tag definitions for that dictionary.

Tag components available in the palette are listed in the following table.

Tag Name	Value	Description
Marker	Marker (default)	Tag name only. The Marker tag does not require a value. The fact that a component has the tag applied is sufficient to convey semantic information (the tag name, that is, "device" or "input").
String	String value	Tag name and string value
Integer	0 (default)	Tag name and numeric value
Long	0 (default)	Tag name and numeric value
Float	0.00 (default)	Tag name and numeric value
Double	0.00 (default)	Tag name and numeric value
Ord	null (default)	Tag name and Ord value
DynamicEnum	0 (default)	Tag name and numeric value
EnumRange		Tag name and numeric value
AbsTime	date/time	Tag name and date/time relative to given time zone
RelTime	00000h 00m 00s	Tag name and time
Unit		Tag name, type and unit of measure
TimeZone	UTC	Tag name and timezone

Figure 27. Example tag properties



Following is a list of common properties for tag (Simple Tag Info) components

Name	Value	Description
Validity		A tag's validity property reflects criteria specified in the Condition property, providing a "hint" as to which objects the tag may be applied.
Condition	Always, And, BooleanFilter,	A validity subproperty used to

Name	Value	Description
	HasAncestor, HasRelation, IsType, Or	specify criteria to be met in order for the tag to be applied to an object. Used in conjunction with the Filter subproperty. There may be one or more conditions under the Validity property.
Filter	A tag name	A Condition subproperty used to further specify objects to which the tag may be applied. The Filter value indicates tag(s) that must already be assigned to an object.
Default Value		Assigns a default value. Often present when validity specifies that the tag may be applied only to a baja:Component .
defFacets		Configurable facets applied to the default value property.

NOTE: Tag dictionaries support adding a single **DataPolicy** to a **TagInfo** or a **TagGroupInfo** component. Also, the **TagInfo** and **TagGroupInfo** components have an **Add DataPolicy** action. For details, see “Data Policies”.

Tag Group Definitions (TagGroupInfoList)

Although not required, a tag dictionary may contain tag groups. The **Tag Group Definitions** folder in a **TagDictionary** property sheet contains the collection of **TagGroups** for that dictionary. A tag group provides a structure that lets you add multiple tags to an object with a single action. Typically, tags are in a group because it is common for each of the tags to be assigned to a single component.

For example, in the Haystack Tag Dictionary, there is a tag group for “discharge air temp sensor” that contains the following set of individual tags:

- discharge
- air
- temp
- sensor

Once a tag group is applied to an object, it implies all of the individual tags in its tag list, as well as implying a marker tag that bears the name of the tag group. This allows you to easily define a NEQL search for the marker tag for that tag group rather than defining a search by concatenating each of the tags in the tag group’s tag list.

The **Device Manager** and **Point Manager** views of a driver, and the **Edit Tags** dialog are the primary methods for adding a Tag Group to a component.

NOTE: When creating or editing a tag group, include only those tags that have a corresponding tag definition in the parent tag dictionary. One way to guarantee it is to populate the tag group’s tag list with tags copied only from the tag definitions list in the tag dictionary.

TagGroup properties

When using the **Edit Tags** dialog to add a tag group, the tag group displays in the **Direct Tags** tab as an **Ord** to the tag group itself, and once the change is saved the individual tags of the tag group display in the **Implied Tags** tab.

A tag group can contain tags from other tag dictionaries. You can add a tag to a tag group that overrides the namespace of the parent tag dictionary. This allows you to define a tag group that contains tags from multiple tag dictionaries.

NOTE: There is no verification that a tag name entered in the **Add Tag** dialog box is actually defined in a tag dictionary. If the tag definition does not exist, the added tag is an ad hoc tag. It is possible to use ad hoc tags, although a tagging best practice is to include only those tags that have a corresponding tag definition in the parent tag dictionary.

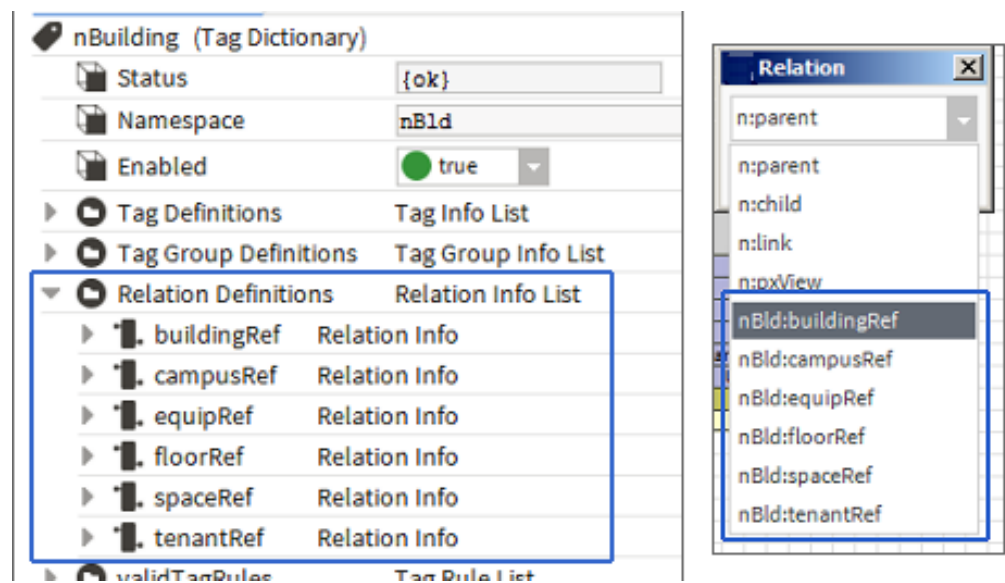
You can add a single “Data Policy” to a **Tag Group Definitions** folder (or to a **Tag Definitions** folder). A data policy provides additional metadata that can be associated with a tagged component. For more details see “Data Policies”.

Type	Value	Description
Validity		Specifies criteria to be met in order for this tag group to be applied to an object.
TagInfoList		Contains the collection of tags that make up this tag group.

Relation Definition (RelationInfo)

Tag dictionaries often contain a collection of relation definitions, which are standardized relation Ids with semantic meaning for that namespace. These relation definitions come into play when adding a relation to a component. In the **Relation** dialog, your choices are limited to the relations that are defined in any of the tag dictionaries installed on your system.

Figure 28. Relation Definitions in custom TagDictionary (left) provide choices seen in the Relation dialog (right)



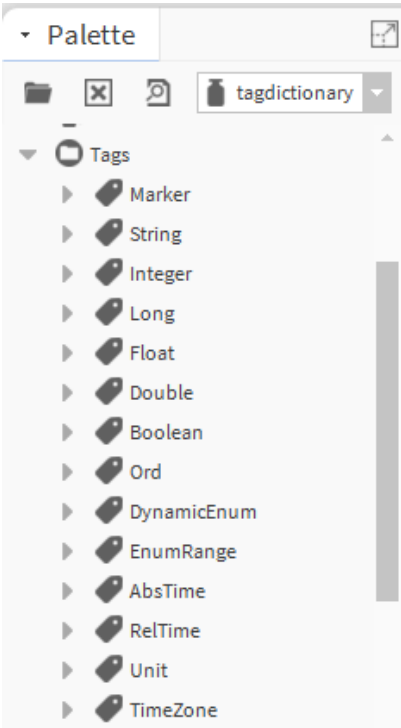
NOTE: For devices containing child points that have a Null Proxy extension, the childNullProxyPoint relation is implied on each of those points.

Tags (SimpleTagInfo)

Several **Simple Tag Info** components, also known as “simple tags” or “tags”, are available in the `Tags` folder of the `tagdictionary` palette. Use these when creating a custom tag dictionary to populate the tag definitions list. Drag these tags from the palette to the **Tag Definitions** folder in the tag dictionary’s property sheet to create the tag definitions for that dictionary.

Tags properties

Tag components and properties are listed below.



Name	Value	Description
Marker	Marker (default)	Tag name only. The Marker tag does not require a value. The fact that a component has the tag applied is sufficient to convey semantic information (the tag name “device” or “input”).
String	text	Tag name and string value
Integer	0 (default)	Tag name and numeric value
Long	0 (default)	Tag name and numeric value
Float	0.00 (default)	Tag name and numeric value
Double	0.00 (default)	Tag name and numeric value
Boolean	true, false (default)	Tag name and boolean value
Ord	null (default)	Tag name and Ord value

Name	Value	Description
DynamicEnum	0 (default)	A DynamicEnum is an ordinal state variable whose range can be specified by an EnumRange.
EnumRange	null (default)	An EnumRange stores a range of ordinal/name pairs.
AbsTime	31-Dec-1969 07:00 AM/PM EST (default)	An AbsTime is an absolute point in time relative to a given time zone.
RelTime	+00000h 00m 00s (default)	A RelTime is relative amount of time.
Unit	micsec() null(null) (default)	Tag name and unit of measure
Timezone	UTC (+0) (default)	A TimeZone value tag specified by an EnumRange.

Smart Tags

The **Smart Tag** components, with the exception of historyMarker and scoped, are already included in the tag lists of rules of the NiagaraTagDictionary, which causes those tags to be implied throughout the station as long as the NiagaraTagDictionary is installed and enabled. While **SimpleTagInfo** components are tags that have just a static value, smart tags are tags whose values are based on some code and the values are usually derived from the objects on which they are applied. A smart tag dictionary includes tag rules that imply tags (both simple tags and smart tags) to components based on the conditions of the rules.

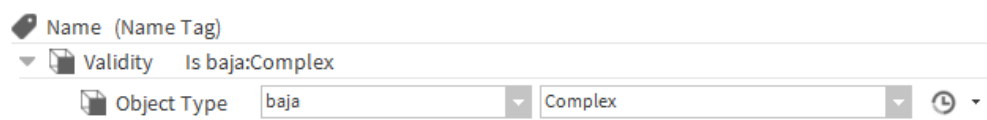
Smart Tag components are available in the tagdictionary palette.

NOTE: Although not recommended, it is possible to replace the NiagaraTagDictionary with your own custom tag dictionary. In that situation, you could drop the smart tag components into the tag list of the rules in your custom tag dictionary so that the tags would be implied throughout your station.

You can add smart tag components to a tag dictionary by dragging them to the tag definitions list, and then drag them onto individual components using the **Edit Tags** dialog box. However, the tags would not be implied, only included in the tag definitions list. To be implied, the smart tags must be included in the tag lists of tag rules. As a rule, these tags are more effectively applied by including them in the tag lists of tag rules.

tagdictionary-NameTag

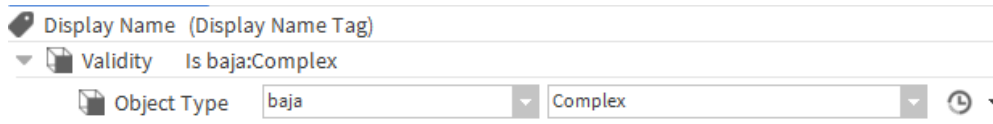
The **Name** smart tag is available in the tagdictionary palette under the **Smart Tags** folder. The value of the name tag is set to the name of the baja:Complex object to which it is applied (either directly or implied by a rule).



Note that the **Name** smart tag is included in the object tags rule of the NiagaraTagDictionary. This rule applies to all objects that are type baja:Complex. The value type of this tag is string.

tagdictionary-DisplayNameTag

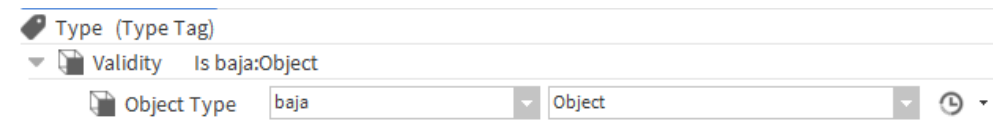
The **Display Name** smart tag is available in the **tagdictionary** palette under the **Smart Tags** folder. The value of the **Display Name** tag is set to the display name of the `baja:Complex` object.



Note that the **Display Name** smart tag is included in the object tags rule of the NiagaraTagDictionary. This rule applies to all objects that are type `baja:Complex`. The value type of this tag is string.

tagdictionary-TypeTag

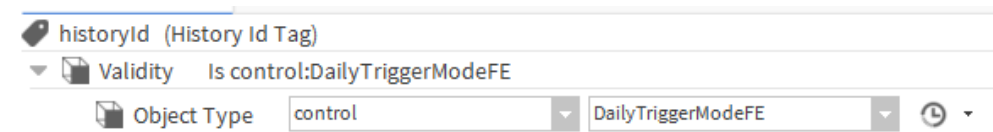
The **Type** smart tag is available in the **tagdictionary** palette under the **Smart Tags** folder. The value of the type tag is set to the type of the `baja:Complex` object.



Note that the **Type** smart tag is included in the object tag rules of the NiagaraTagDictionary. This rule applies to all objects that are type `baja:Complex`. The value type of this tag is string.

tagdictionary-historyIdTag

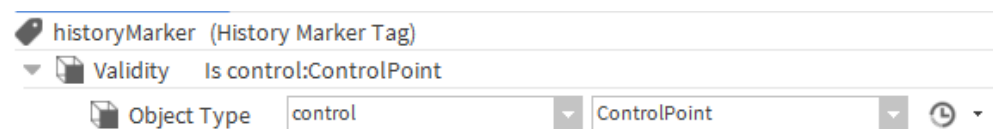
The **historyId** smart tag is available in the **tagdictionary** palette under the **Smart Tags** folder.



The **historyId** smart tag is included in the point tags rule of the NiagaraTagDictionary. This rule applies to all objects that are type `control:ControlPoint`. The value type of this tag is string and its value is set to the historyId of the first enabled history extension on the `control:ControlPoint` object.

tagdictionary-HistoryMarkerTag

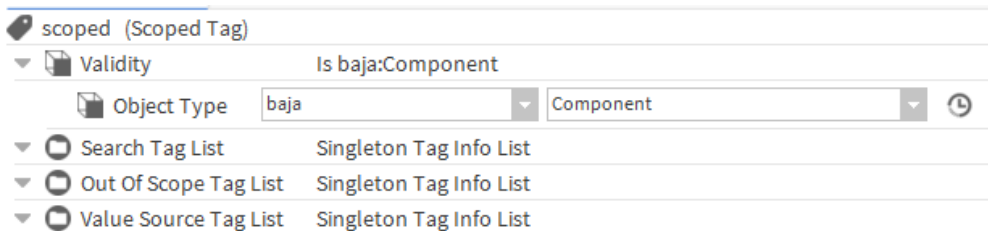
The **historyMarker** tag is available in the **tagdictionary** palette under the **Smart Tags** folder. The **historyMarker** smart tag, if included in a rule, is a marker tag that is applied to a `control:ControlPoint` object if it has an enabled history extension.



tagdictionary-ScopedTag

The **scoped** smart tag is available in the tagdictionary palette under the **Smart Tags** folder. The **scoped** smart tag can be applied to a `baja:Component` object.

Figure 29. Scoped smart tag



When applied to a component, either directly or implied by a tag rule, the smart tag searches the component's ancestors looking for a tag with the "search ID". This tag is called the matching tag and the ancestor on which it is found is called "the matching ancestor".

Without specifying anything in the scoped tag's frozen properties, the basic behavior of this smart tag is as follows:

- The search ID will be the same as scoped tag itself.
- If found, a tag with the scoped tag ID will be added to the component. The value of the added tag will be copied from the matching tag. If the matching tag is a marker tag, the added tag will be a marker tag. If the matching tag is a value tag, the added tag will have the same value as the matching tag.

This basic behavior can be modified by adding a **TagInfo** to the **scoped** tag's frozen properties. Usage is optional and each can contain only a single tag. Only the ID of the **TagInfo** added to these properties is of consequence and its type is ignored.

- **Search Tag List property**

If a **TagInfo** is added to **Search Tag List** property, the ID of this **TagInfo** will be used as the search ID instead of the ID of the scoped tag. If a matching tag is found, the tag added to the component will still have the same ID as the scoped tag. If this property is empty or the **TagInfo** has the same ID as the scoped tag, only direct tags will be considered when searching the component's ancestors. Otherwise, both direct and implied tags will be considered.

- **Out Of ScopeTag List property**

If a **TagInfo** is added to the **Out Of Scope Tag List** property, the ID of this **TagInfo** is called the "out-of-scope ID". The scoped tag will not be added to the component if the component itself, the matching ancestor, or any ancestors between the component and the matching ancestor have a tag with the out-of-scope ID. If the out-of-scope ID is the same as the scoped tag ID, only direct tags will be considered. Otherwise, both direct and implied tags will be considered. If the out-of-scope ID is the same as the scoped tag ID, the scoped tag will never be added to a component.

- **Value Source Tag List property**

If a **TagInfo** is added to the **Value Source Tag List** property, the ID of this **TagInfo** is called the "value-source ID". If a tag exists on the matching ancestor with the value-source ID, called the "value-source tag", the value of the added tag will be copied from that tag instead of the matching tag. If a value-source tag does not exist on the matching ancestor, the value of the added tag will be copied from the matching tag. If the value-source ID is the same as the scoped tag ID, only direct tags will be considered for the value-source tag. Otherwise, both direct and implied tags will be considered.

For example, if the tag with the search ID is a double value tag set to 5.0, then a double value tag set to 5.0 with the scoped tag ID will be added to descendant components. If a **TagInfo** is added to the **ValueSourceTagList** property, the value of the scoped tag added to a component will be copied from the direct or implied tag with the same ID as the **ValueSourceTagList** TagInfo (the value source ID) on the matching ancestor. Only direct tags (and not implied tags) will be used if the **ValueSourceTagList** TagInfo has the same ID as the scoped tag. If the matching ancestor does not have a tag with the value source ID, the value of the scoped tag will be copied from the tag with the search ID.

In another example using the namespace of the custom dictionary is "c:". The **Search Tag List** is populated so c:stateRoot will be used as the search ID instead of hs:geoState, which is the ID of the scoped tag. The **Out Of Scope Tag List** is empty. The **Value Source Tag List** is populated so the value-source ID is n:name. If a direct or implied tag with the ID c:stateRoot is found on an ancestor of a component, a hs:geoState tag will be added to that component with a value set to the n:name tag on the matching ancestor.

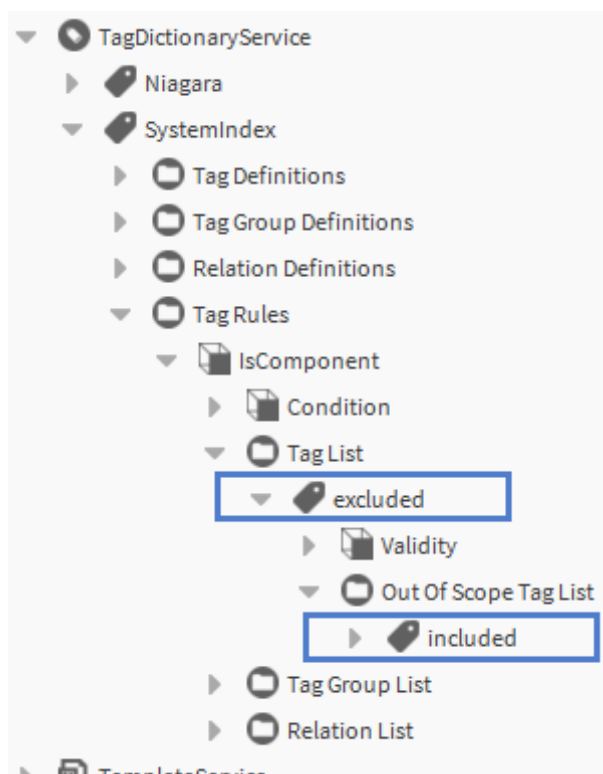
Figure 30. Example scoped tag: hs:geoState

hs:geoState (Scoped Tag)		
▶	Validity	Is baja:Component
▼	Search Tag List	Singleton Tag Info List
▶	stateRoot	Marker
▶	Out Of Scope Tag List	Singleton Tag Info List
▼	Value Source Tag List	Singleton Tag Info List
▶	n:name	Marker

SystemDb usage example

Provides a usage example for the scoped smart tags in the SystemIndex tag dictionary.

Note that the excluded scoped smart tag is available in the tag rules of the SystemIndex tag dictionary. When a systemIndex:excluded marker tag is added to a component, a systemIndex:marker tag becomes implied on all of the component's descendants except those that contain or are descendants of a component with a systemIndex:include marker tag. The presence of the systemIndex:excluded tag, whether direct or implied, excludes the component from system indexing operations. Alternately, to prevent implying the excluded tag on some of the descendants you can apply the included tag where necessary.

Figure 31. Excluded and included smart tags in SystemIndex dictionary

This usage example shows a components tree where a direct `systemIndex:excluded` marker tag is added to A1 and a direct `systemIndex:included` marker tag is added to B12. If the SystemIndex tag dictionary is installed, the resulting implied `systemIndex:excluded` tags are shown in parentheses. Due to the `systemIndex:included` (the OutOfScope id) tag on B12, a `systemIndex:excluded` marker tag is not implied on B12 or its descendants (C121 and C122).

Figure 32. SystemDB included/excluded example

```

* station root
  * A1: systemIndex:excluded
    * B11: (systemIndex:excluded)
      * C111: (systemIndex:excluded)
      * C112: (systemIndex:excluded)
    * B12: systemIndex:included
      * C121
      * C122
    * B13: (systemIndex:excluded)
      * C131: (systemIndex:excluded)
      * C132: (systemIndex:excluded)
  * A2
    * B21
      * C211
      * C212
    * B22
      * C221
      * C222

```

If A1 contains a direct or implied systemIndex:excluded tag and the OutOfScope id (as shown below), the systemIndex:excluded tag is not added to any of A1's descendants.

Figure 33.

```

* station root
  * A1: systemIndex:excluded systemIndex:included
    * B11
      * C111
      * C112
    * B12: systemIndex:included
      * C121
      * C122
    * B13
      * C131
      * C132
  * A2
    * B21
      * C211
      * C212
    * B22
      * C221
      * C222

```

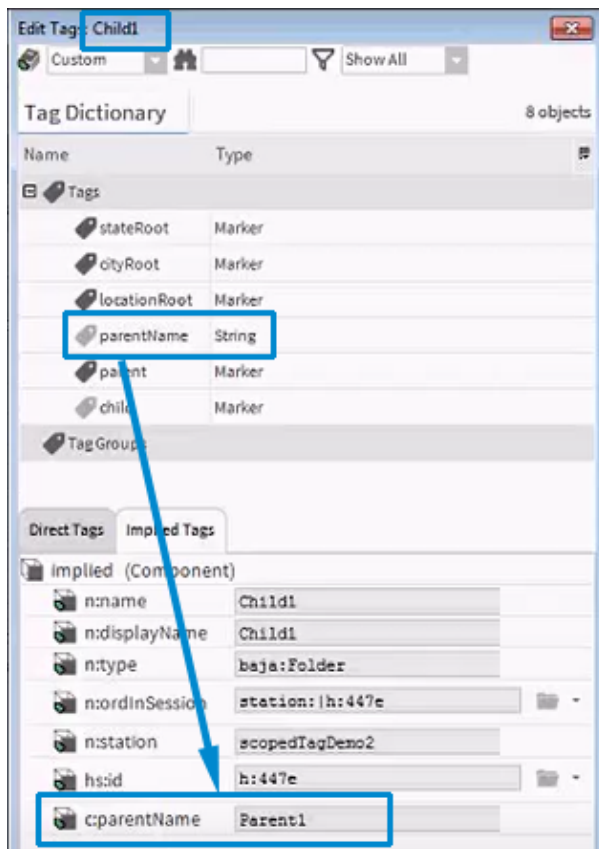
Hierarchy QueryLevelDef usage example

Provides a usage example for scoped smart tags in a hierarchy QueryLevelDefinition.

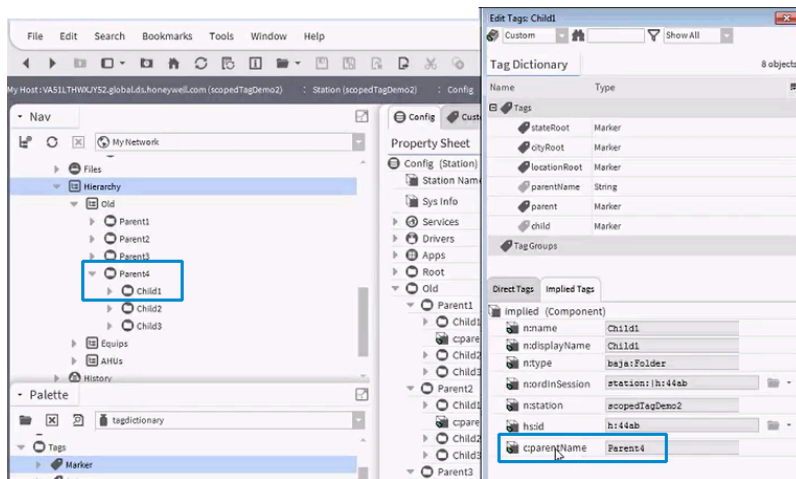
To add a c:parentName tag with the value of the n:name tag to all descendants of a component with a

"c:parent" tag, create a scoped tag with the ID c:parentName, set the **Search Tag List** property to c:parent, and set the **Value Source Tag List** property to n:name.

Figure 34. parentName becomes an implied tag on the descendant



You can use this by setting up a hierarchy to first query for all objects in the station that have a c:parent tag, and then using a query facet, to query for all objects in the station that have both a c:child tag and a c:parentName tag that is the same name as the parent component.

Figure 35. Hierarchy utilizing scoped tags

When defining a hierarchy, one advantage to using `QueryLevelDefs` rather than `GroupLevelDefs` is that you can attach a graphic view to a `QueryLevelDefinition`. On expanding the resulting hierarchy, the graphic view displays for that level.

tagdictionary-SingletonTagInfoList

The **Singleton Tag Info List** is a frozen property in a `ScopedTag` smart tag component, available in the `tagdictionary` palette under the Smart Tags folder. Usage is optional.

Each **Singleton Tag Info List** has a particular use, and each can contain only a single tag. For more details, see [tagdictionary-ScopedTag](#)

tagdictionary-TagRuleList

The **Tag Rules** used in a Smart Tag Dictionary are the primary mechanism for implying tags and relations. It is the tag rules defined in the installed tag dictionaries that determine which implied tags and implied relations are assigned to each object (entity).

A tag rule defines certain criteria that determines if one or more tags and/or relations are implied on an object. In addition to the `Condition` property, tag rules contain three definition lists: Tag List, Tag Group List, and Relation List.

When a “tag-able” object is evaluated, the process determines if the object meets the criteria specified in the `Condition` property of each tag rule. If the criteria is met, it will return a tag (or relation, or tag group) with the value set (if other than a Marker tag). If the criteria is not met, then the implied tag does not apply and a null value is returned. Eventually, the results from the tag rules in all of the smart tag dictionaries in the station are merged to form the complete set of implied tags and implied relations for an object.

The `validity` slot of a definition (`TagInfo`, `RelationInfo`, or `TagGroupInfo`) is not evaluated in a tag rule or in a tag group definition. It is only evaluated in the Tag Definitions of a tag dictionary.

Any definition (tag, tag group, or relation) that exists in a Tag Rule or Tag Group Definition is required to have a corresponding definition in the main lists of the tag dictionary (Tag Definitions, Tag Group Definitions, Relations Definitions).

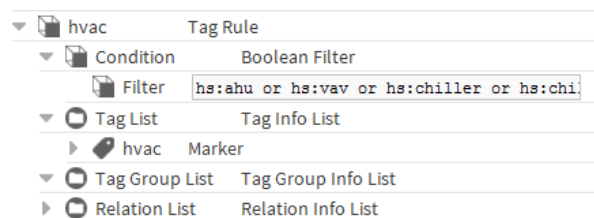
Added scoped tag rules provide a means of focusing tag rule evaluation in specific areas of the station tree.

Tag Rule components

The tagdictionary palette contains the default TagRule, TagForType which has the IsType condition, however the **Condition** slot is frozen, uneditable, limiting its usage. It also contains additional Rule components allowing you to add custom TagRules (with different Conditions) to smart tag dictionaries. These new TagRule components in the palette cover all possible conditions (except Never), which means that tags can be implied using more complex logic.

For additional information, see “tagdictionary-TagRuleList” in this guide.

Figure 36. Example tag rule (Haystack Tag Dictionary)



The tag rule shown here is the BooleanFilterRule. The rule's **Condition** slot has a **Filter** field containing the NEQL predicate which queries for the following tags: `hs:ahu or hs:vav or hs:chiller or hs:chillerPlant or hs:coolingTower or hs:heatExchanger or hs:boiler or hs:boilerPlant`. Also, the rule's **Tag List** is configured with the `hvac` marker tag. This tag rule queries objects in the station, filtering for those that have one or more of the tags specified in the condition **Filter** field. If the query returns `true` for any object (as having one or more of the queried tags), then the rule applies the `hvac` implied tag to the object as well.

Each of the TagRule components is configured for a certain condition. If these conditions are met (present in station objects) then the specified tag(s) and/or relation(s) become implied for those objects.

tagdictionary-ScopedTagRule

Tag rules may have a scope in which they apply. This means that an entity will only have tags implied by a rule if the entity is in the tag rule's scope. The effect of this is to focus evaluation of NEQL queries on applicable entities, which may reduce the amount of time it takes to complete a search or to perform hierarchy traversal. Several **Scopes** components are available in the tagdictionary palette.

Regular tag rules do not have a scope container in which to drop scope components, so they cannot be scoped. The special ScopedTagRule type has a **Scope List** container in which scopes may be dropped from the palette. A scoped tag rule is targeted to a particular place in the station via the **Scope Ord** property. Also, within a custom smart tag dictionary, you can create multiple scoped tag rules, each with a different root **Scope Ord**. Moreover, a single ScopedTagRule may have multiple scopes defined in its scope container — if an entity is in any of these scopes, the rule applies.

NOTE: For tag rules that are not targeted to a particular place in the station, it is better to use a “regular” TagRule rather than a Scoped Tag Rule because the scoping mechanism requires some time to complete the evaluation so it may slow performance if everything is in scope.

The reason to use a scoped tag rule is to speed up certain kinds of tag rule evaluations. In particular, using a `hasAncestor` condition to specify a particular part of a station (such as, `hasAncestor` with `n:name = Building1Folder`) in a tag rule is slow and this new scoping mechanism speeds up those kinds of rules. However, if the rule is something that does not require checking an entity's ancestors, then a regular TagRule should be used instead.

For example, suppose you are using a rule to tag components by their name, say a rule with a BooleanFilter with `n:name` like `"Lighting.*East"`, and you know that those components will only live in a particular part of the station, say under a folder called `"Campus"`. You might think that using a scoped tag rule would make the

queries faster because it would not have to check outside the Campus folder. That is not the case though. It still has to check every component and decide if it is in scope or not. In this case, determining if a component is in scope is probably slower than just checking the name of the component, so using the scoped rule could hurt performance slightly.

Basically, the scoped tag rule exists as an alternative to using the slower `hasAncestor` condition to check for a component's scope in the station. A best practice is to consider if you did not use a scoped tag rule, would you use a `hasAncestor` condition to accomplish the same thing. If so, use a scoped tag rule. If not, use a regular `TagRule`.

AlwaysRule

If the `AlwaysRule` is present in an installed tagdictionary, the condition is `true` for all station objects and so the specified tag(s) may be applied to each and every object.

Name	Value	Description
Condition	true (default)	There is no criteria to be met. The condition is always "true" so a filter is not necessary.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

AndRule

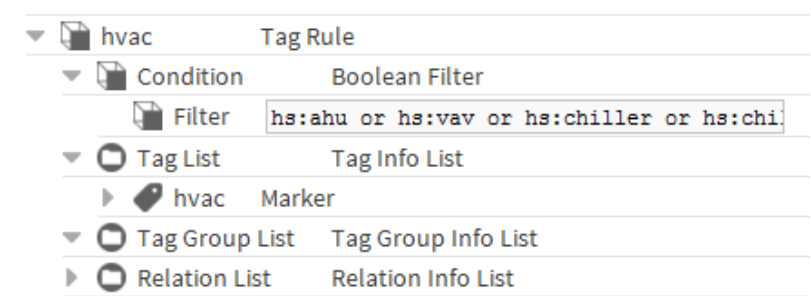
The specified tag definitions may be applied if the target object has *all* of the conditions listed under the `Condition` property.

Name	Value	Description
Condition	true (default)	There is no criteria to be met. The condition is always "true" so a filter is not necessary.
Condition sub-properties	tagdictionary: conditionName null (default)	Add slots under Condition property or drag any of the Condition components (Always, And, BooleanFilter, HasAncestor, HasRelation, IsType, Never, Or) from the palette.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

BooleanFilterRule

The specified tags may be applied if the target object has a tag listed in the `Filter` property.

Figure 37. Example BooleanFilter tag rule



Name	Value	Description
Condition		Criteria to be met by target objects.
Condition sub-properties	baja:stringfalse (default)	Target object must meet the filter criteria (NEQL query).

Name	Value	Description
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

HasAncestorRule

The specified tag definition(s) may be applied if the target object, or one of its ancestors, has the tags listed in the **Filter** property.

Name	Value	Description
Condition		Criteria to be met by target objects.
Filter	baja:stringfalse (default)	Target object or one of its ancestors must meet the filter criteria (NEQL query).
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Info List contains the list of tag(s) to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

HasRelationRule

The specified tag definition(s) may be applied if the target object, or an object along the relation specified in the **Relation Id** field, has the tags listed in the **Filter** property and has the relation specified in the **Relation Id** field.

Name	Value	Description
Condition		Criteria to be met by target objects.
Filter	baja:stringfalse (default)	Target object must meet the filter criteria (NEQL query).
Relation Id	baja:string	Relation along which objects will be searched for one that meets the Filter criteria.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

IsTypeRule

The specified tag definition(s) may be applied if the target object is one of the specified object type.

Name	Value	Description
Condition		Criteria to be met by target objects.
Is Type Condition (sub-property)	baja:TypeSpec	Target object must be or extend this specified type.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

NotRule

The specified tag definition(s) may be applied if the child condition of the Not tag rule condition evaluates to false. The tag definition(s) will not be applied if the Not tag rule condition has no child condition.

Figure 38. NotRule tag rule

NotRule (Tag Rule)

Condition

Not

IsType

Is null

Object Type

Tag List

Tag Info List

Tag Group List

Tag Group Info List

Relation List

Relation Info List

Name	Value	Description
Condition (Not)		Criteria to be met by target objects.
Child Condition (sub-property)	tagdictionary:TagRuleCondition	Target object must be or extend this specified type. The NotRule condition only accepts a single child tag rule condition. The IsType condition is shown but the child condition can be any subclass of BTagRuleCondition.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

OrRule

The specified tag definition(s) may be applied if the target object has *any* of the conditions listed under the **Condition** property.

Figure 39. Example OrRule tag rule

OrRule (Tag Rule)

Condition

Or

Cond1

Is control:ControlPoint

Object Type

control

ControlPoint

Cond2

Has Ancestor

Filter

hs:ahu

Tag List

Tag Info List

Name	Value	Description
Condition		Criteria to be met by target objects.
Condition sub-properties	tagdictionary: conditionName null (default)	Add slots under Condition property or drag any of the Condition components (Always, And, BooleanFilter, HasAncestor, HasRelation, IsType, Never, Or) from the palette.
Tag List	tag(s)	Tag Info List contains the list of tag(s) to be applied.
Tag Group List	tag group(s)	Tag Group Info List contains the list of tag groups to be applied.
Relation List	relation Id(s)	Relation Info List contains the relation Ids to be applied.

Conditions

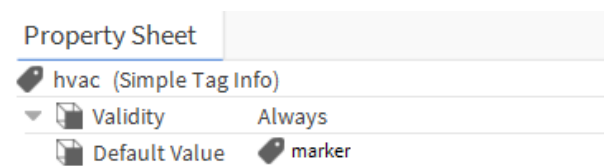
Condition is a validity subproperty used to specify criteria to be met in order for a parent definition's tag to be applied to an object. It is used in conjunction with the **Filter** subproperty. There may be one or more conditions under the Validity property.

The **Validity** property of a tag definition is primarily used to filter out tags in the **Edit Tags** dialog that should not be applied to an object. The criteria contained in this property does not prevent applying the tag to any object. Some smart tags, however, do not operate properly if they are applied to an object that does meet the criteria in the **Validity** property. The **Validity** property can be any tag rule condition including **And** and **Or**, which allow for combining other conditions.

The validity conditions are located in the **tagdictionary** palette.

tagdictionary-Always

Always is one of several possible validity conditions properties. If the **Always** condition property is present in a tag definition, there is no criteria to be met. The condition is always **true** and so the parent definition's tag(s) and/or relation(s) become implied for every object in the station.

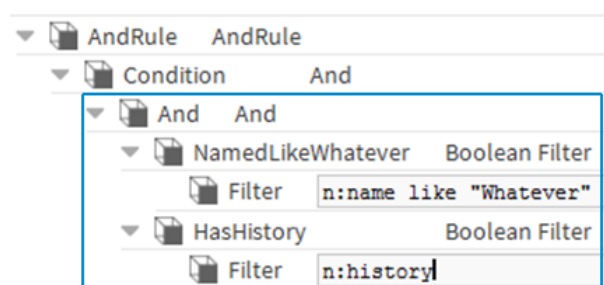


The validity conditions are located in the **tagdictionary** palette.

tagdictionary-And

And is one of several possible validity conditions properties. If all of the specified criteria are present in the queried object, the parent definition's tag(s) may be applied.

For example, you can drag an **AndRule** onto the rule definitions of a smart tag dictionary. Then, drag the **And** condition (or other conditions) under the condition **And** for that rule, as shown here.

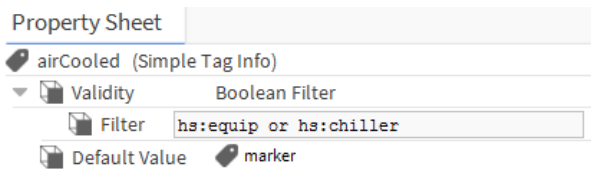


The validity conditions are located in the **tagdictionary** palette.

tagdictionary-BooleanFilter

BooleanFilter is one of several possible validity conditions subproperties, which are used to specify a certain criteria to be met in order for a tag to be applied to an object.

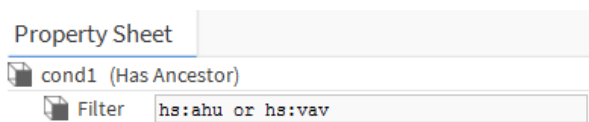
For example, the **BooleanFilter** condition slot has a **Filter** subproperty containing the NEQL predicate which queries objects in the station for the tags: `hs:equip` or `hs:chiller`. If the query returns `true` for any object (as having one of more of the queried tags), the validity criteria has been met. So the parent definition's tag(s) and/or relation(s) become implied for those objects.



tagdictionary-HasAncestor

HasAncestor is one of several possible TagRule conditions, which are used to specify a tag rule or tag validity criteria.

Figure 40. Example HasAncestor Condition slot with Filter



In the example shown, the **HasAncestor** condition slot has a **Filter** subproperty containing the NEQL predicate which queries ancestor entries in the station: `hs:ahu` or `hs:vav`. If the **Filter** NEQL predicate is `true` for the target object or one of its ancestors, the **HasAncestor** criteria is met. If an object is a component and has a parent component, its ancestor is that parent. If an object is not a component or is a component without a parent component, the endpoint of the first outbound `n:parent` relation is its ancestor.

tagdictionary-HasRelation

HasRelation is one of several possible TagRule conditions which are used to specify a tag rule or tag validity criteria.

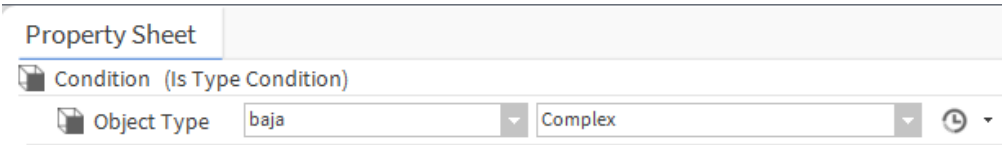
The **Filter** subproperty contains a NEQL predicate. If this predicate is `true` for the target object or a related object, the **HasRelation** criteria is met. A related object is the endpoint of any inbound or outbound relations where the relation **Id** matches the one specified in the **Relation Id** field. Endpoints of the endpoints, endpoints of those endpoints, and so on are also tested.

If the target object, or an object along the relation specified in the **Relation Id** field, has the tags listed in the **Filter** property and has the relation specified in the **Relation Id** field, then the parent definition's tag(s) may be applied.

tagdictionary-IsType

IsType is one of several possible validity conditions subproperties, which are used to specify a certain criteria to be met in order for a tag to be applied to an object.

If the queried object is of the specified object type, then the parent definition's tag(s) may be applied.



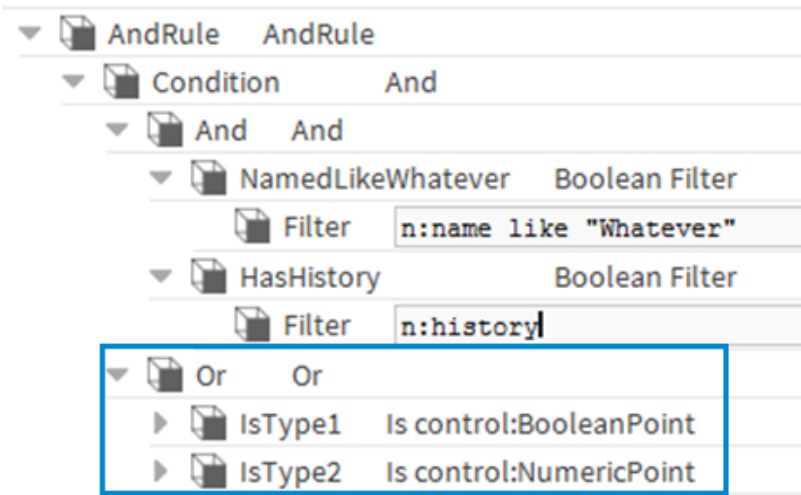
The validity conditions are located in the **tagdictionary** palette.

tagdictionary-Or

Or is one of several possible validity conditions subproperties, which are used to specify a certain criteria to be met in order for a tag to be applied to an object.

For the **Or** condition, if the target object has any of the conditions listed under the **Or** property, the parent definition's tag(s) may be applied.

The example below shows the **And** tag rule which contains the condition **And** with these additional combined conditions: **And** and **Or**. In this case, if all of the **And** criteria plus any of the **Or** criteria are met, the parent definition's tag(s) may be applied.



tagdictionary-OrdScope

You add the **OrdScope** component to the scope list container or a **ScopedTagRule**. Added **OrdScopes** are, in effect, adding another condition to the rule.

Properties

NOTE: In order for the rule to be applied, the component must be located at or under the **scopeOrd** and not located under any of the **excludedScopeOrds**.

The **OrdScope** components are available in the **tagdictionary** palette.

Property	Value	Description
Scope Ord	ord, null (default)	Defines the part of the station tree to

Property	Value	Description
		evaluate. For example, during a search or hierarchy refresh the <code>DriverContainerScope</code> ord value, <code>service:driver:DriverContainer</code> , limits evaluation to objects in the Drivers folder. Similarly, the <code>NiagaraNetworkScope</code> ord value, <code>service:niagaraDriver:NiagaraNetwork</code> limits evaluation to objects in the <code>NiagaraNetwork</code> folder.
Exclude Scope Ords	Ord List	Defines part of the station tree excluded from evaluation. For example, during a search or hierarchy refresh this exclude scope ords value, <code>service:TagDictionaryService</code> , excludes the Tag Dictionary Service from evaluation. Typically, the ords in this ord list will be underneath the scope ord.

tagdictionary-DataPolicy

Tag dictionaries may include data policies. A data policy provides additional metadata associated with a tag or tag group. The tagdictionary palette contains the following **DataPolicy** components:

- DataPolicy
- BooleanDataPolicy
- EnumDataPolicy
- NumericDataPolicy
- StringDataPolicy

NOTE: Typical tagging operations do not require data policies. The data policy functionality is provided primarily for use by the Analytics engine. For that reason, tag dictionaries may include added data policies.

An **Add Data Policy** action has been added to **TagInfo** and **TagGroupInfo** components. Invoking this action prompts you to select a DataPolicy type to add to the selected **TagInfo** or **TagGroupInfo** component in a tag dictionary.

You can also add a data policy to **TagInfo** or **TagGroupInfo** components by dragging a **DataPolicy** component from the tagdictionary palette onto the desired **TagInfo** or **TagGroupInfo** component.

NOTE: Only a single data policy can be added to **TagInfo** or **TagGroupInfo** components. You may not add a data policy if the **TagInfo** component is a Marker tag, or if the **TagInfo** or **TagGroupInfo** component already has a DataPolicy child.

Properties for Data Policy

Name	Value	Description
Min Interval	drop-down with time intervals (defaults to <code>None</code>)	Defines the minimum allowed interval when requesting a value or trend request. When a value or trend request is processed, if the value of the <code>Interval</code> in the request is less

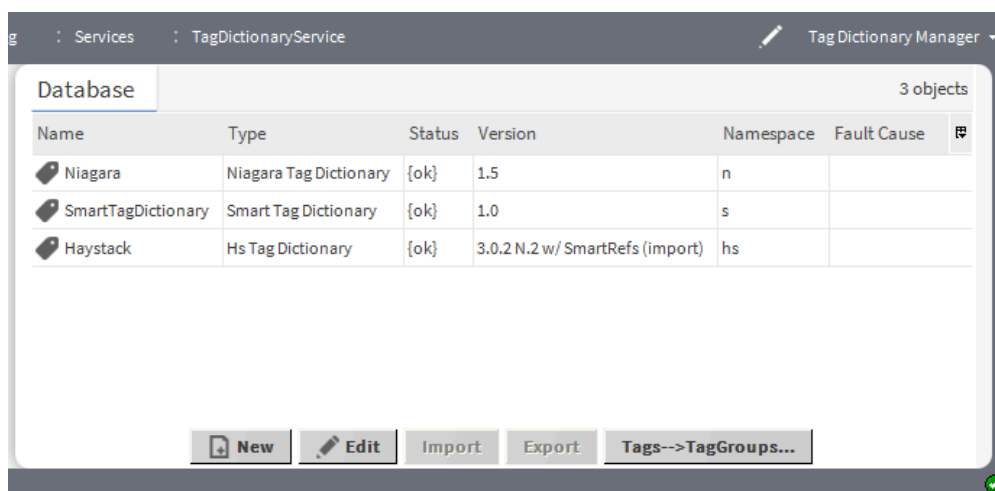
Name	Value	Description
		<p>than the value of this Min Interval, the Algorithm modifies the value of the Interval in the request to be the value of this Min Interval prior to processing the request.</p> <p>For example, if a trend request specifies the Interval as Five Minutes and the algorithm's Min Interval is Fifteen Minutes the algorithm changes the request Interval from Five Minutes to Fifteen Minutes before processing the request.</p> <p>The default of None allows any interval to be specified in the request.</p>
Max Interval	drop-down with time intervals (defaults to None)	<p>Defines the maximum allowed Interval when requesting a trend.</p> <p>When an algorithm processes a value or trend request, if the value of the request's Interval is greater than the value of this Max Interval, the algorithm modifies the value of the request's Interval to the value of this Max Interval prior to processing the request.</p> <p>For example, if the trend request specifies an Interval of Week and the algorithm's Max Interval is Day the algorithm changes the value of the request's Interval from Week to Day before processing the request.</p> <p>The default of None allows any Interval to be specified in the request.</p>
Preferred Time Range	from and to times (both default to 12 AM EDT)	Configures when the time period starts and ends.
Preferred Rollup	drop-down list of arithmetic functions	Defines the how to roll up the data when a request does not specify the rollup function.

Name	Value	Description
Preferred Aggregation	drop-down list of arithmetic functions	Defines the aggregation to use when a request does not specify the aggregation function.
Units	drop-down list of units of measure	Identifies which unit of measurement system to use, English or metric. EnglishMetricNone
Precision	32 bit (default), 64 bit	Selects 32 bit or 64 bit options for the history data logging. The 64 bit option allows for higher level of precision but consumes more memory.
Totalized		
Trend Required	true (default) or false	When true, all points must have a trend.

Tag Dictionary Manager view

Tag Dictionary Manager, the default view for the Tag Dictionary Service, lists all tag dictionaries installed on the station and their versions. The view provides functionality for creating, editing, importing/exporting tag dictionaries; and indicates the presence direct tags that could be replaced with a TagGroup relation.

Figure 41. Tag Dictionary Manager view



Buttons

- New** — allows you to create a new smart tag dictionary, which you can then export to a .CSV file for editing.
- Edit** — at the bottom of the view allows you to change the tag dictionary properties: Name, Namespace,

and Enabled status.

- **Import and Export** — allow you to import or export tag dictionaries in a standard .CSV file format. You can edit an exported tag dictionary in any CSV-compatible spreadsheet program (either online or offline) to add or remove tag definitions, tag groups definitions, relation definitions, as well as validity rules. Afterwards, you may import the edited .CSV file, thereby updating the existing tag dictionary.
- **Tags > TagGroups** — examines the station component tree looking for individual direct tags on a component that match a TagGroup defined in an installed tag dictionary. If any are found, a window opens listing the component, collection of direct tags, matching TagGroup and an indication if the component is contained in a deployed template. Also shown are two columns, **Convert** and **RemoveTags**.

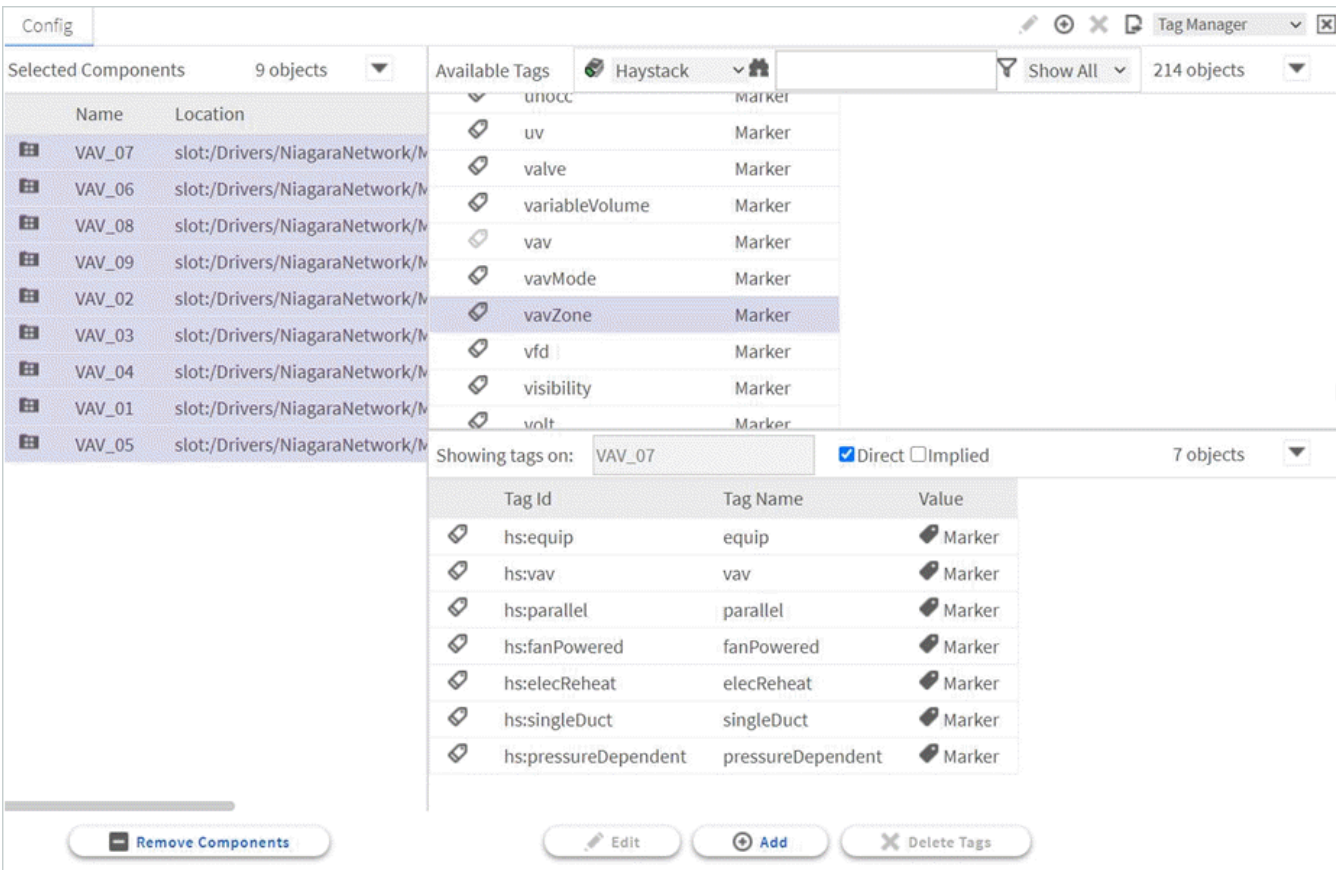
If the **Convert** column is checked, the direct tags will be replaced with a TagGroup relation. When Convert is selected RemoveTags is automatically be selected. If the Convert is not selected but **RemoveTags** is, the tag collection will be removed from the component without converting to a TagGroup relation. This is to handle the case where a collection of tags can map to multiple TagGroups with some being subsets of others. If neither Convert or RemoveTags is selected, no action is taken.

Make Convert and RemoveTag selections in the presented table and when you click **OK**, the selected Conversions and TagRemovals occur.

HTML5 Tag Manager view

There is added support for the **HTML5 Tag Manager** view featuring an intuitive design that enables a more efficient overall tagging workflow.

Figure 42. HTML5 Tag Manager view in a browser



It provides the same functionality as the **Edit Tags** window in Workbench, but with some additional workflow

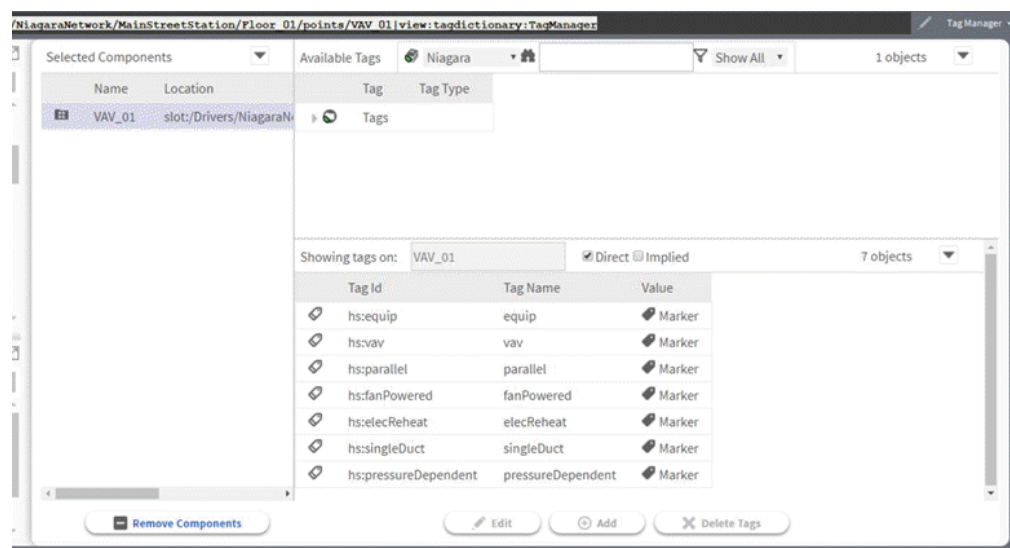
enhancements. For example, the “context sensitive tagging” feature (as shown in the above image), is also available in the existing **Edit Tags** window via the dropdown list with either the **Show All**, **Best Only** or **Valid Only** option; and provides advanced tag filtering as well.

The following enhancements in the **HTML5 Tag Manager** result in a more efficient tagging workflow:

- Allows bulk tagging of components via the browser
- Seamlessly view direct and implied tags
- Export all direct and implied component tags to a spreadsheet for templating and ongoing management
- Integrates with the **Search Service** to improve workflow
- Drag-n-drop functionality
- Added as a default view on all station components for enhanced flexibility

The manager view functions identically whether accessed from within Workbench or a browser.

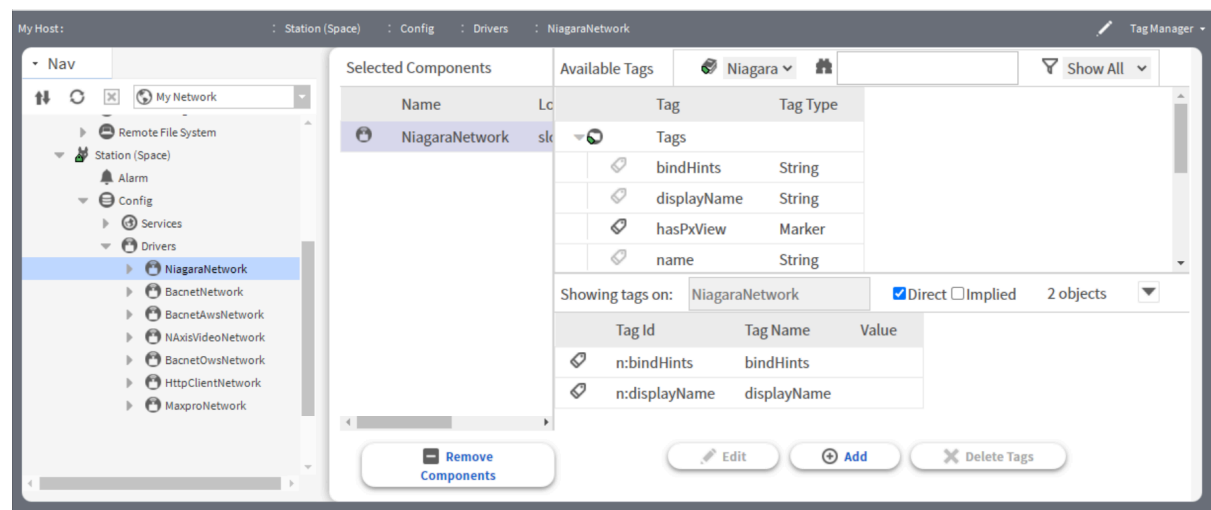
Figure 43. HTML5 Tag Manager view in Workbench



Tag Manager view

This view is the default view of the **Tag Manager**. You can add bulk tags.

Figure 44. Tag Manager View



To access this view, right-click on any network container and click **Views > TagManager**.

The following sections explains about the different types of windows.

Selected Components

Column	Description
Name	Display the name of the component.
Location	Display the location of the selected component.

Available Tags

Column	Description
Tag	Display the different types of tags names.
Tag type	Display the type of tag.

Showing tags on (Direct/Implied)

There are two types of tags available

- **Direct Tags** Direct tags are tags that you add intentionally to a component using an installed tag dictionary.
- **Implied Tags** Implied tags are tags that are not directly stored in the component, but are "implied" by tag rules that are defined in installed Smart Tag Dictionaries.

Column	Description
Tag Id	Display the tag Id.
Tag Name	Display the tags name.
Value	Display the tag value.

Column	Description
Value Type	Display the type of value.

Buttons

- **Remove Components** Removes the selected component from the database
- **Edit**
Edit the value of the tag.
- **Add** Insert a new tag in the database.
- **Delete** Deletes a selected tag from the database.

Relation Manager view

This view is an HTML5 view that you can use to add, edit, and delete relations and links (as of Niagara 4.15). It is available as a view on all components.

Nav

Config

Services

Drivers

ComponentsFolder

- NumericWritable
- NumericWritable1
- tempProgram
- NumericWritable2
 - AMS_NumericWritable

Files

Histories

Palette

- alarm
- baja
- control
- docDeveloper
- history
- jsonToolkit
- kitControl
- kitPx
- nSnmpp
- schedule
- tagdictionary
- testTagDictionary

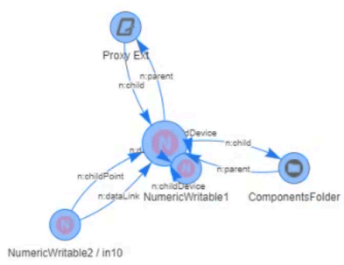
NumericWritable

Show Direct

Show Implied

Show Links

Show Model



Outbound	Relation	Relation	Inbound	Tags
NumericWritable2 / in10	Link	NumericWritable / out		
		NumericWritable	n:childDevice	NumericWritable1
NumericWritable1	n:childDevice	NumericWritable		
NumericWritable2	n:childPoint	NumericWritable		
		NumericWritable	n:parent	Proxy Ext
ComponentsFolder	n:parent	NumericWritable		
		NumericWritable	n:child	ComponentsFolder
Proxy Ext	n:child	NumericWritable		
		NumericWritable / in16	Link	NumericWritable1 / out

New

Edit

Delete

The following section explains the different properties.

Property	Description
Outbound	Displays the component name of an outbound relation or link (includes the slot for links)
Relation	Displays the (outbound) relation ID for a relation or the type for a link
Component	Displays the name of the component at which this view is looking (includes the slot for links)
Relation	Displays the (inbound) relation ID for a relation or the type for a link
Inbound	Displays the component name of an inbound relation or link (includes the slot for links)
Tags	Opens a compact editor to add, edit, or remove tags on a link

Actions

- **New:** Adds a relation or link
- **Edit:** Edits the selected relation or link
- **Delete:** Deletes the selected relations and/or links

Chapter 5. Glossary

The following glossary entries relate specifically to the topics that are included as part of this document. To find more glossary terms and definitions refer to glossaries in other individual documents.

Alphabetical listing

semantic information

Metadata used to indicate the purpose of a device, that is, what the device is, what each of its data points means, and how devices are related to each other.

scoped tag rule

In Niagara tag rules have a scope in which they apply. This means that an entity will only have tags implied by a tag rule if the entity is within the tag rule's scope. This focuses evaluation of NEQL queries on applicable entities, which reduces the amount of time it takes to complete a search or hierarchy refresh.

data policy

A data policy provides additional metadata that can be associated with a tagged component. For more details on tags and tagging, see the Niagara Tagging Guide.

Haystack

An extensible Semantic Web Browser developed by the Haystack research group at the MIT Computer Science and Artificial Intelligence Laboratory (<http://haystack.lcs.mit.edu>). The project explores how the Semantic Web data model (a Resource Description Framework — RDF) can be applied by users to better organize, navigate, and retrieve information, both personal and shared ([www.w3.org/2005/04/swls/BioDash/Demo/What is Haystack.html](http://www.w3.org/2005/04/swls/BioDash/Demo/What%20is%20Haystack.html)).

Haystack tag dictionary

A smart tag dictionary (namespace) containing a collection of tags developed by Project Haystack, which can be used for semantic modeling of building control entities, i.e. site tags, building tags, equipment tags, point tags, geo-location tags, etc.

The Haystack dictionary is indicated by the *hs* character, followed by a colon character (:).

The Haystack dictionary is a result of the work of the Open Source Initiative (OSI) hosted on the website <http://project-haystack.org>.

implied tags index

In Niagara you can manually enable indexing on individual tags in the tag definitions of a custom smart tag dictionary. The index primarily improves performance of NEQL searches and speeds-up hierarchy refreshing.

namespace

A container for a set of names in a naming system. A tag dictionary is a namespace.

Niagara tag dictionary

A tag dictionary (namespace) containing a collection of tags developed for Niagara systems, that are used for semantic modeling of specific building control entities, i.e. networks, devices, equipment, points, sites, buildings, geo-location, histories, etc.

The Niagara dictionary is a type of Smart Tag dictionary, therefore it applies Implied Tags and Implied Relations to components and links. This allows queries to find these components based on type, linkage, hierarchy or combinations of these. The Niagara tag dictionary is included by default in all stations created using the New Station tool.

The Niagara dictionary is indicated by the *n* character, followed by a colon character (:).

tag

A piece of semantic information (metadata) associated with a device or point (entity) for the purpose of filtering or grouping entities. Tags identify the purpose of the component or point and its relationship to other entities. For example, you may wish to view only data collected from meters located in maintenance buildings as opposed to those located in office buildings or schools. For this grouping to work, the metering device in each maintenance building includes a tag that associates the meter with all the other maintenance buildings in your system.

JACEs are associated with Supervisors based on tags; searching is done based on tags.

Tags are contained in tag dictionaries. Each tag dictionary is referenced by a unique namespace.

tag dictionary

Tag dictionaries contain a set of tag definitions, and may contain tag group definitions, relation definitions, as well as tag rules for smart tags.

taggable spaces

The implementation of tags for all common data types: components, files, histories and alarms.

tag rule index

In Niagara the tag rule index is an index on the station's Tag Dictionary Service which improves performance in evaluating tag rules for implied tags during NEQL searches.