HighByte IntelligenceHub User Guide 4.3

Source: guide.highbyte.com

(10/30/2025)

Table of Contents

1. User Guide
2. 3.4 User Guide | User Guide
3. Central Configuration | User Guide
4. Compare Hubs | User Guide
5. Network Group | User Guide
6. Remote Hub | User Guide
7. Sync Hubs | User Guide
8. Tagging | User Guide
9. Configuration | User Guide
10. Administration | User Guide
11. Certificates | User Guide
12. Environment Variables | User Guide
13. Project | User Guide
14. Export | User Guide
15. Import | User Guide
16. Project API | User Guide
17. SAML and Active Directory | User Guide
18. Secrets | User Guide
19. System Variables | User Guide
20. Users and Roles | User Guide
21. Condition | User Guide
22. Aggregate | User Guide
23. Alarm | User Guide
24. Custom | User Guide
25. Deadband | User Guide
26. Connect | User Guide
27. Connections | User Guide
28. AI | User Guide
29. Amazon Bedrock | User Guide
30. Azure OpenAI | User Guide
31. Google Gemini | User Guide
32. OpenAI | User Guide
33. Amazon | User Guide
34. Redshift | User Guide
35. S3 | User Guide
36. S3 Tables | User Guide
37. Data Firehose | User Guide

38. IoT SiteWise | User Guide
39. Kinesis Data Streams | User Guide
40. Microsoft Azure | User Guide
41. Blob Storage | User Guide
42. Event Hubs | User Guide
43. IoT Edge | User Guide
44. IoT Hub | User Guide
45. OneLake | User Guide
46. Microsoft Fabric | User Guide
47. Databricks Client | User Guide
48. Databricks SQL | User Guide
49. Files | User Guide
50. CSV | User Guide
51. File | User Guide
52. Parquet | User Guide
53. Google | User Guide
54. BigQuery | User Guide
55. PubSub | User Guide
56. Historians | User Guide
57. AspenTech IP.21 | User Guide
58. AspenTech IP.21 Agent | User Guide
59. InfluxDB | User Guide
60. PI System | User Guide
61. PI System AF SDK Connector | User Guide
62. TimescaleDB | User Guide
63. Ignition Module | User Guide
64. Installing Module | User Guide
65. Kafka | User Guide
66. MCP Client | User Guide
67. Modbus TCP | User Guide
68. MQTT | User Guide
69. OPC UA TCP | User Guide
70. OPC-UA CSV Import | User Guide
71. Troubleshooting | User Guide
72. REST Client | User Guide
73. Snowflake SQL | User Guide
74. Snowflake Streaming | User Guide
75. Sparkplug | User Guide
76. SQL | User Guide
77. JDBC Driver | User Guide
78. Microsoft SQL Server | User Guide
79. MySQL | User Guide
80. Oracle Database | User Guide
81. PostgreSQL | User Guide
82. SQLite | User Guide
83. Webhook | User Guide
84. Inputs | User Guide
85. Outputs | User Guide
86. Model | User Guide

87. Instances | User Guide
88. Expressions | User Guide
89. AI Generate Instances | User Guide
90. Model Import | User Guide
91. Models | User Guide
92. Namespace | User Guide
93. Migrate Parameters | User Guide
94. Other | User Guide
95. Functions | User Guide
96. Output Templates | User Guide
97. Parameters | User Guide
98. References | User Guide
99. Templating | User Guide
100. Pipeline | User Guide
101. Stages | User Guide
102. Common | User Guide
103. Breakup | User Guide
104. Filter | User Guide
105. Flatten | User Guide
106. Model | User Guide
107. Model Validation | User Guide
108. Size Buffer | User Guide
109. Time Buffer | User Guide
110. Control | User Guide
111. Delay | User Guide
112. Error | User Guide
113. For Each | User Guide
114. On Change | User Guide
115. Return | User Guide
116. Subpipeline | User Guide
117. Switch | User Guide
118. While | User Guide
119. Format | User Guide
120. CSV | User Guide
121. GZip | User Guide
122. JSON | User Guide
123. Parquet | User Guide
124. Zip | User Guide
125. I/O | User Guide
126. Merge Read | User Guide
127. Read | User Guide
128. Smart Query | User Guide
129. Write | User Guide
130. Write New | User Guide
131. Parse | User Guide
132. CSV | User Guide
133. JSON | User Guide
134. Parquet | User Guide
135. XML | User Guide

136. Transform | User Guide
137. JavaScript | User Guide
138. JSONata | User Guide
139. State | User Guide
140. Triggers | User Guide
141. API | User Guide
142. Callable | User Guide
143. Cron | User Guide
144. Event | User Guide
145. Flow | User Guide
146. Polled | User Guide
147. Troubleshoot | User Guide
148. Debug | User Guide
149. Error Handling | User Guide
150. Replay | User Guide
151. Statistics | User Guide
152. Servers | User Guide
153. MCP Services | User Guide
154. UNS Client | User Guide
155. Setup | User Guide
156. Application Settings | User Guide
157. Command Line Options | User Guide
158. Docker | User Guide
159. Git Integration | User Guide
160. Git Example | User Guide
161. High Availability | User Guide
162. Installation | User Guide
163. Application State | User Guide
164. Installing Java | User Guide
165. Licensing | User Guide
166. OpenTelemetry | User Guide
167. Redundancy | User Guide
168. System Requirements | User Guide
169. Upgrade | User Guide
170. Upgrading from 3.4 to 4.0 | User Guide
171. Event.metadata | User Guide
172. Event.value | User Guide
173. Troubleshooting | User Guide
174. Dashboard | User Guide
175. Debug Logging | User Guide
176. Event Log | User Guide
177. Debugging Pipelines | User Guide
178. Status & Metrics | User Guide

User Guide

> Source: https://guide.highbyte.com/

Intelligence Hub Guide

Welcome to the HighByte Intelligence Hub User Guide.

Setup

- Setup
  - Application Settings
  - Command Line Options
  - Docker
  - Git Integration
  - Installation
  - High Availability
  - OpenTelemetry
  - Redundancy
  - System Requirements
  - Upgrade
- Configuration
  - Connect
  - Model
  - Pipeline
  - Namespace
  - Condition
  - UNS Client
  - Administration
  - Servers
  - Other
- Troubleshooting
  - Event Log
  - Status & Metrics
  - Dashboard
  - Debug Logging
  - Debugging Pipelines
- Central Configuration
  - Network Group
  - Remote Hub
  - Tagging
  - Compare Hubs
  - Sync Hubs
- 3.4 User Guide

Troubleshooting

- Setup
  - Application Settings
  - Command Line Options
  - Docker
  - Git Integration
  - Installation
  - High Availability

    - OpenTelemetry
    - Redundancy
    - System Requirements
    - Upgrade
- Configuration
    - Connect
    - Model
    - Pipeline
    - Namespace
    - Condition
    - UNS Client
    - Administration
    - Servers
    - Other
- Troubleshooting
    - Event Log
    - Status & Metrics
    - Dashboard
    - Debug Logging
    - Debugging Pipelines
- Central Configuration
    - Network Group
    - Remote Hub
    - Tagging
    - Compare Hubs
    - Sync Hubs
- 3.4 User Guide

Configuration

- Setup
    - Application Settings
    - Command Line Options
    - Docker
    - Git Integration
    - Installation
    - High Availability
    - OpenTelemetry
    - Redundancy
    - System Requirements
    - Upgrade
- Configuration
    - Connect
    - Model
    - Pipeline
    - Namespace
    - Condition
    - UNS Client
    - Administration
    - Servers
    - Other
- Troubleshooting
    - Event Log
    - Status & Metrics
    - Dashboard
    - Debug Logging
    - Debugging Pipelines
- Central Configuration
    - Network Group

    - Remote Hub
    - Tagging
    - Compare Hubs
    - Sync Hubs
- 3.4 User Guide

3.4 User Guide | User Guide

> Source: https://guide.highbyte.com/34_user_guide/

3.4 User Guide

See the link below to access the 3.4 guide.

- 3.4 User Guide

Central Configuration | User Guide

> Source: https://guide.highbyte.com/central_configuration/

Central Configuration

The Intelligence Hub supports the ability for any hub to act as a central hub, allowing for the central hub to act as a single interface for configuring multiple remote hubs. This is useful when a single facility has many Intelligence Hub instances running closer to the machines or in cases where it is easier to manage many Intelligence Hubs centrally, either by application or facility.

Enable Central Configuration

To enable central configuration mode for a specific Intelligence Hub instance, enable the `Enable Central Config` setting in the *Admin* | *Settings* configuration.

Once enabled, two new options will appear in the side panel:

- Host which allows changing hubs
- Hubs which allows managing connected child hubs.

More information about Central Config

- Network Group
- Remote Hub
- Tagging
- Compare Hubs
- Sync Hubs

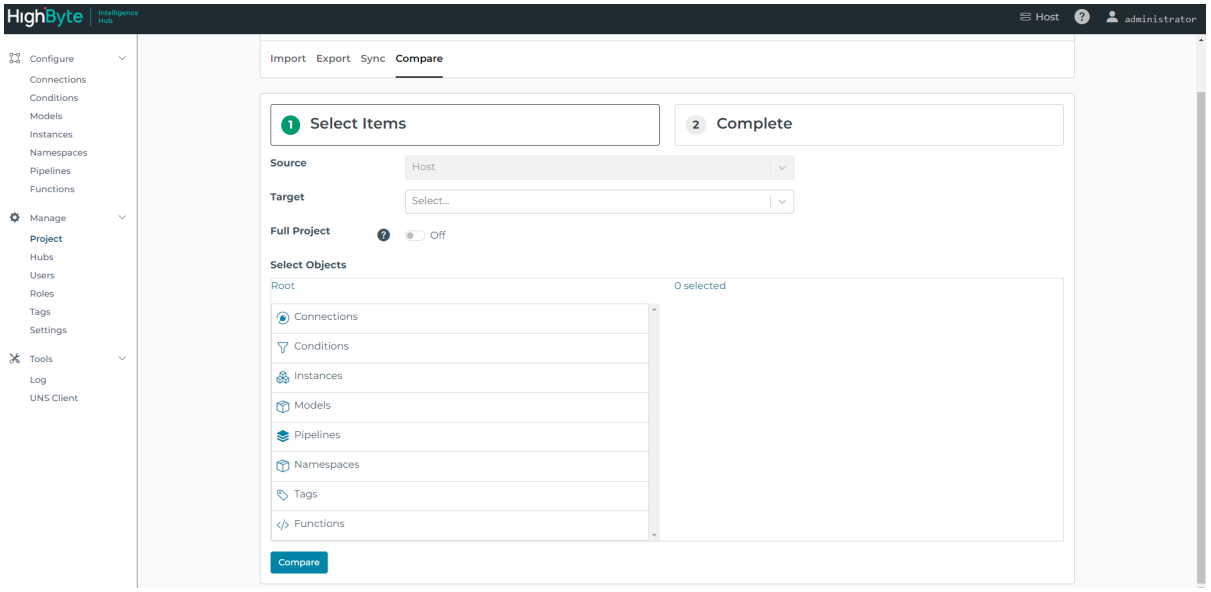
Compare Hubs | User Guide

Source: https://guide.highbyte.com/central_configuration/compare_hubs/

Compare Hubs

Central Configuration can be used to quickly compare individual configuration objects or entire projects between two hubs. To compare objects, perform the following steps.

1. Click *Project* and select the *Compare* tab.
2. The *Source* hub is the current hub you are configuring.
3. The *Target* is the hub being compared to.
4. Select *Full Project* to compare everything to the target. If *Full Project* is disabled, use the browse and select feature to select Tags or individual objects to compare. When selecting a Tag, all objects with that Tag are selected. You can then de-select any objects that you don't wish to compare.
5. Click *Compare*.
6. The *Complete* page shows the results of the comparison, including objects that are different or missing on the target. To see the difference between the object on the source and the object on the target hub, users can expand each element. Users can also select a set of elements using the checkboxes and click the **Sync Selected** button to pre-populate the sync page with the selected elements and the target.



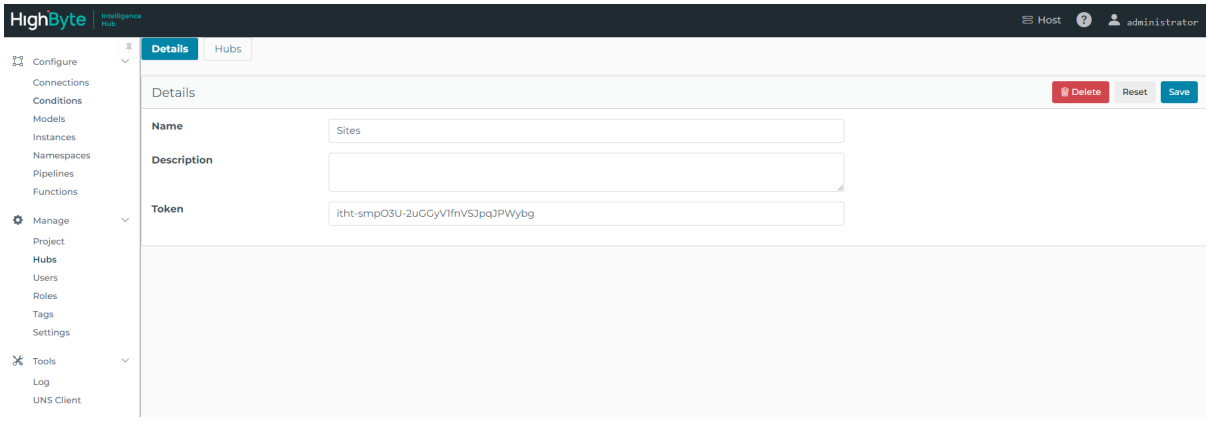
Network Group | User Guide

> Source: https://guide.highbyte.com/central_configuration/network_group/

Network Group

A network group represents a group of remote hubs and contains the credentials a remote hub needs to connect to the central hub. To create a network group, perform the following steps.

1. *Click Network* and select the *Hubs* tab to list all *Network Groups* previously configured.
2. *Click* the *Create Network Group* button to create a new group.
3. *Enter* a Name for the group and an optional description.
4. *Click* the *Save* button for the change to take effect.
5. A *Token* will be automatically created for the *Network Group*. This *Token* will be required by a corresponding remote Intelligence Hub for authentication with this central hub.



Remote Hub | User Guide

> Source: https://guide.highbyte.com/central_configuration/remote_hub/

Remote Hub

Connect a Remote Hub

Once a Network Group is created, perform the following steps to connect a remote hub to the central hub.

The remote hub connects to the central hub's REST API (default port 45245) using websockets. The central hub needs to expose a websockets port to allow remote hubs to connect.

1. *Install* the Intelligence Hub *runtime component* on another machine. Only the runtime is required, but you may optionally install the configuration component to allow for direct configuration if desired. See section Intelligence Hub Runtime Installation.
2. *Create* a new `intelligencehub-remoteconfig.json` file in the appData directory. Define the file as follows. An `intelligencehub-remoteconfig.json.template` file is provided in the runtime directory as a starting point.

```
{
  "remoteUrl": "ws://hostname:45245/websocket",
  "token": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "clientId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

3. `remoteUrl` is the URL to the central host's REST API. By default, this is port 45245 and uses web sockets (ws://) over HTTP or secure web sockets (wss://) over HTTPS.

   Note: *HighByte recommends the use of HTTPS for accessing the REST API for secure websocket communications*

4. `token` is the token automatically generated when creating a Network Group on the central hub and is used to authenticate the remote hub.
5. `clientId` is the unique identifier used for the remote hub when connecting to the central hub. It can be configured by the user or left blank to be auto-generated by the central hub. If the specified client id doesn't exist on the central hub, a new network hub will be created on the central hub to represent the remote hub. The clientId must be a GUID.
6. *Save* this file and *restart* the runtime for this remote hub installation. The remote hub will connect to the central host.

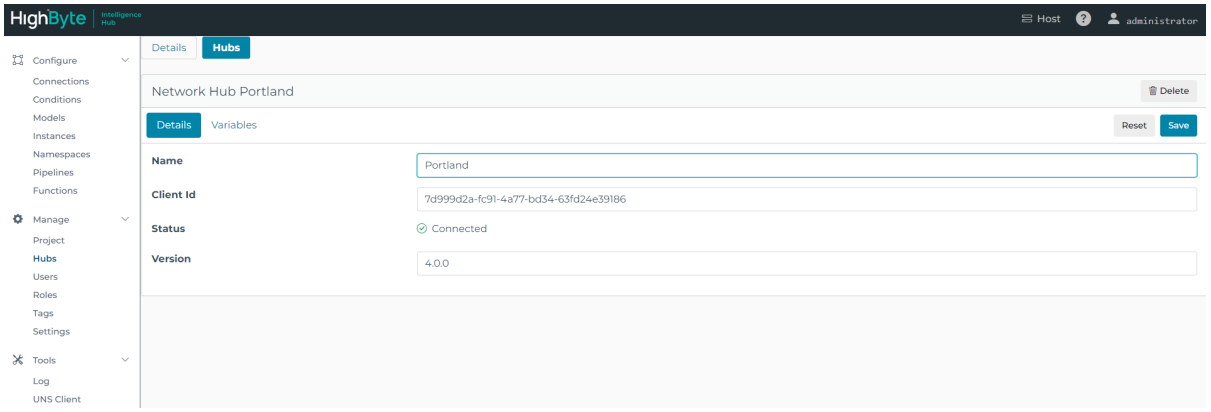
   Upon successfully connecting, the following event message will be emitted: `Remote configuration connected to ws://hostname:45245/websocket`.

Rename the Remote Hub

By default, the remote hub connects to the central hub and uses its machine name as an identifier. This identifier can be changed by performing the following steps.

1. Click *Network* and select the *Hubs* tab to list all Network Groups previously configured.
2. Select the corresponding *Network Group* in the list and *View* its *Details*.
3. Click *Hubs* to list all currently connected Hubs.
4. Rename the hub and click *Save*.

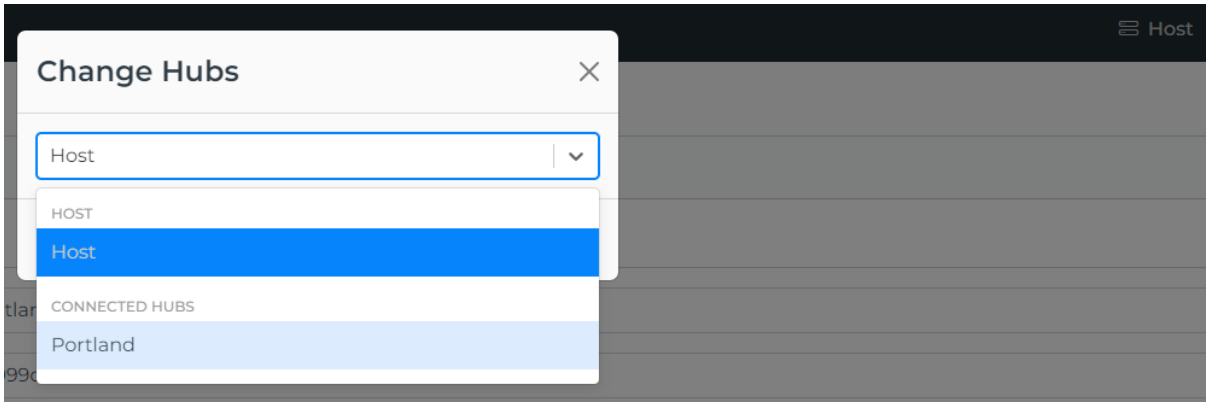
The hub's *Connected State* and its Client ID is also displayed. The Client ID is automatically assigned to the remote hub when it first connects. On future connections, the remote hub provides the Client ID to identify the hub.



Remotely Configure the Hub

To configure the connected remote hub, perform the following steps.

1. Click *Host* from the main menu.
2. Select the remote hub to configure in the *Change Hubs* dialog and click *Update*.
3. All configuration changes will now be proxied to the remote hub. When finished, select another remote hub to configure or click *Host* to work with the central hub.



Sync Hubs | User Guide

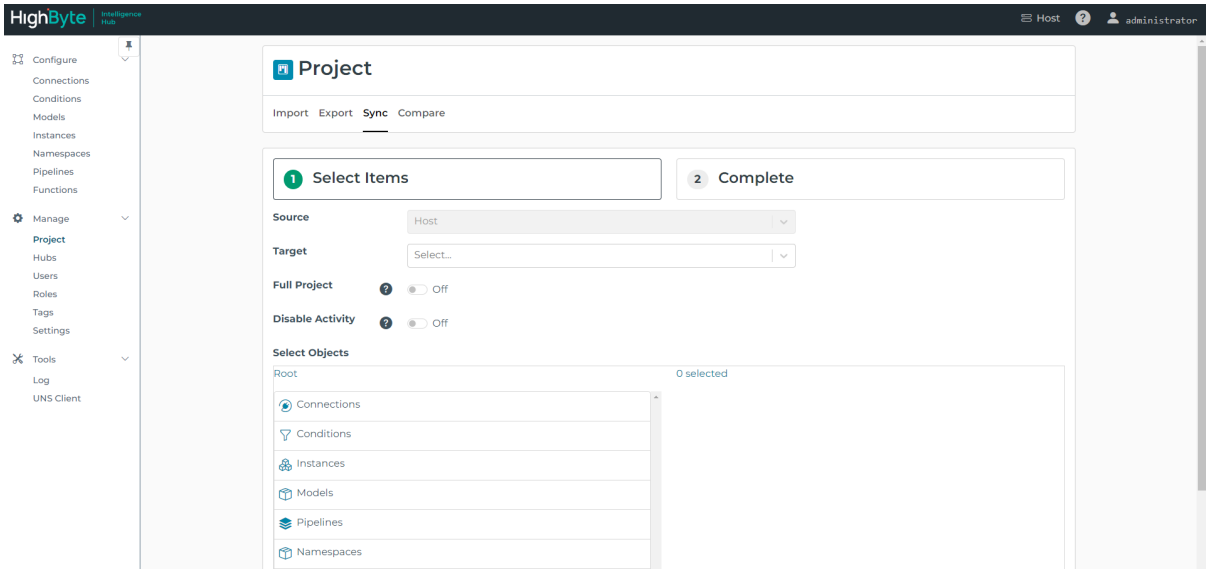
Source: https://guide.highbyte.com/central_configuration/sync_hubs/

Sync Hubs

Central Configuration can be used to quickly synchronize individual configuration objects or entire projects between two hubs. To synchronize objects, perform the following steps.

1. Click *Project* and select the *Sync* tab.
2. The *Source* hub is the current hub you are connected to.
3. The *Target* is the hub being synchronized.
4. To sync everything to the target, select *Full Project*. If *Full Project* is disabled, use the browse to select individual objects or tagged objects. When selecting a Tag, all objects with tagged with that Tag are selected. You can then de-select any objects that you don't wish to sync.
5. To sync all pipelines as inactive (triggers disabled), enabled the *Disable Activity* setting. Otherwise, pipelines are synced with the current trigger enabled/disabled state.
6. Click *Sync*.
7. The *Complete* tab shows whether synchronization succeeded for each object.

Synchronization cannot be rolled back or reverted and partial success is possible if some objects fail to synchronize.



Tagging | User Guide

Source: https://guide.highbyte.com/central_configuration/tagging/

Tagging

Tagging allows configuration objects (Connections, Conditions, Models, Instances, and Flows) to be grouped by use case, common components, or arbitrary groups. Once grouped, the UI can be filtered to only show objects with certain tags. This allows a large project to be broken up by use case.

Example

For example, a user may have components used as part of an OEE (Overall Equipment Effectiveness) use case, and another group of objects used for an Environmental use case. To take this further, both use cases may have common cloud connectors, and these cloud Connections could be tagged with “Cloud” in addition to the use case tag.

Once objects are tagged, the UI can filter on “OEE” and only see configuration for the OEE use case.

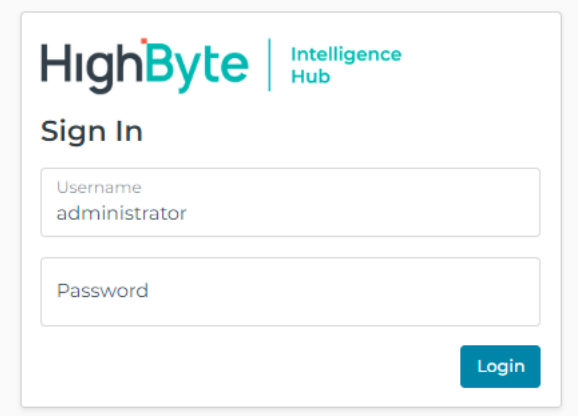
In Central Configuration tags allow configuration objects to be grouped and managed as a whole. For example, one can compare the “OEE” tag across the host and remote hub, to see if there are any changes. Optionally, users could then choose to re-deploy the “OEE” tag to update the use case on the remote hub. Users could also use tagging to update the credentials for “Cloud” tagged connectors and push those credential updates to the remote hubs.

Configuration | User Guide

> Source: https://guide.highbyte.com/configuration/

Configuration

By default, HighByte Intelligence Hub Configuration is available via the browser at the following address http://localhost:45245.



By default, the username is `administrator` with no password. See the Users and Roles for setting up passwords and limiting access.

The following sections contain information on how to configure various features of HighByte Intelligence Hub.

- Connect
  - Connections
  - Inputs
  - Outputs
- Model
  - Models
  - Model Import
  - Instances
- Pipeline
  - Triggers
  - Stages
  - State
  - Troubleshoot
- Namespace
  - Migrate Parameters
- Condition
  - Aggregate
  - Alarm
  - Custom
  - Deadband
- UNS Client
- Administration
  - Users and Roles
  - System Variables
  - Environment Variables
  - Secrets
  - Certificates
  - SAML and Active Directory

    - Project
- Servers
    - REST Data Server
    - MCP Services
- Other
    - References
    - Parameters
    - Templating
    - Functions
    - Output Templates

Administration | User Guide

> Source: https://guide.highbyte.com/configuration/administration/

Administration

This section covers the various administrative functions available in the Intelligence Hub.

- Users and Roles
- System Variables
- Environment Variables
- Secrets
- Certificates
- SAML and Active Directory
- Project
  - Export
  - Import
  - Project API

Certificates | User Guide

Source: https://guide.highbyte.com/configuration/administration/certificates/

Certificates

Certificates can be viewed under Settings. Certificates are used to encrypt and validate connections to other systems (Ex. connecting via TLS to and MQTT broker or OPC UA server). This section covers how to create and manage certificates.

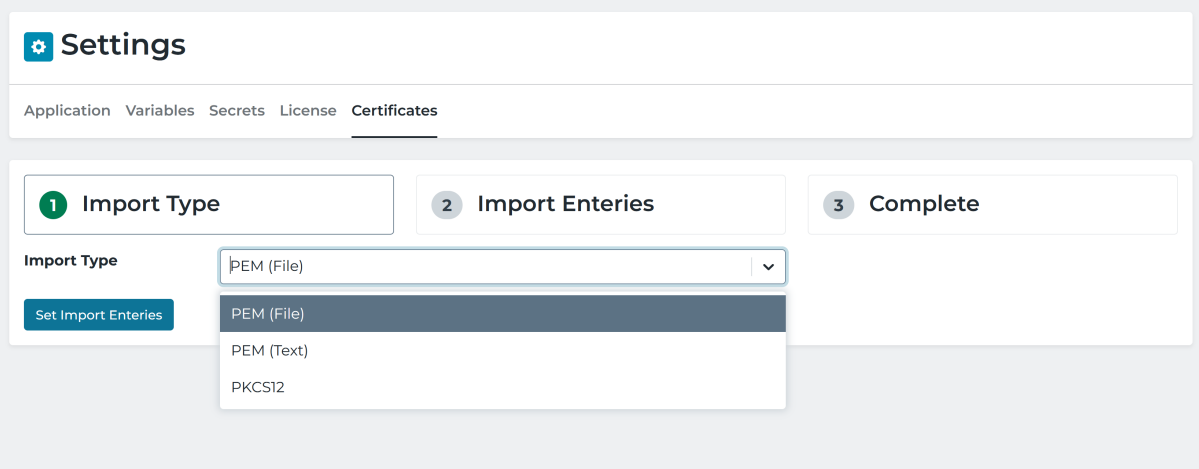
Certificates are commonly used to secure communications and authenticate clients. An example is using certificates with the MQTT connector to send and receive data from AWS IoT Core. In this case, the self-signed certificate for AWS is used to secure the connection and AWS provides a public and private key to authenticate the hub. Use the steps in the preceding sections to import certificates into the hub and use them in connectors.

Add a Certificate

1. Navigate to **Settings** > **Certificates** in the configuration's Main Menu. Here you can add, edit, and view certificates. To add a certificate, click the **New Certificate** button.

   NOTE: HighByte Intelligence Hub may automatically generate some certificates/keys on your behalf (e.g., `app-certificate*` are application instance-specific certificates used by OPC UA connections). You may replace these with your own certificates and keys according to your organization's IT policies.

2. Choose the upload format you want to use:
   - **POM**: Supports both text and file-based uploads of certificates and keys.
   - **PKCS12**: Supports upload of `.p12` or `.pfx` bundled certificate/key files.



1. For **POM** uploads, choose how you'd like to provide your certificate and key:

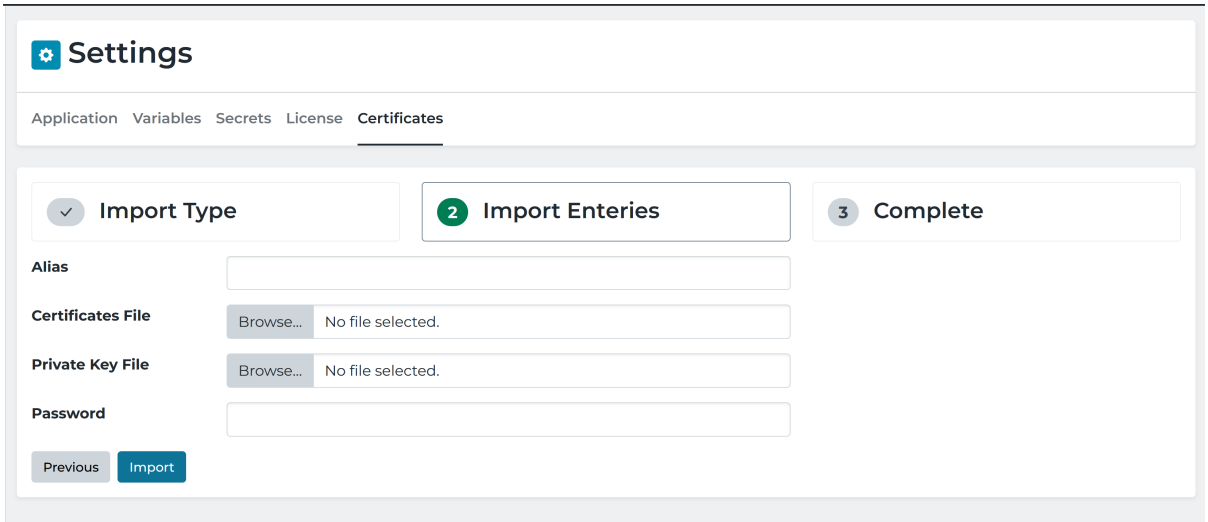
   - **Text**: Paste the contents of your certificate and private key directly into the provided fields.
   - **File**: Upload individual certificate and key files.

   Supported formats include:

   - **X.509 certificates**
   - Private keys in **PKCS#1** or **PKCS#8** formats

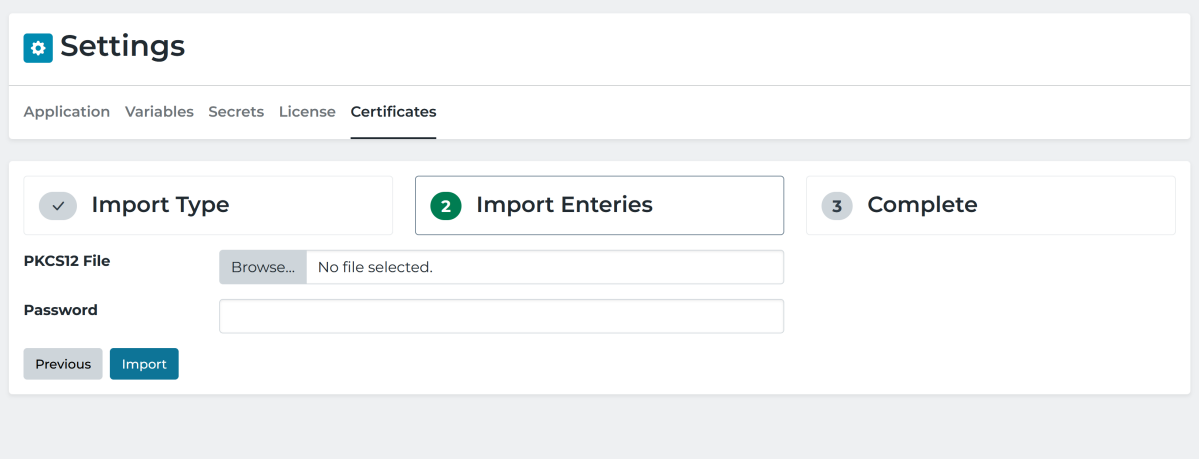
   Example:

   - Certificates should begin with `-----BEGIN CERTIFICATE-----`.
   - Private keys should start with either `-----BEGIN RSA PRIVATE KEY-----`, `-----BEGIN PRIVATE KEY-----`, or `-----BEGIN ENCRYPTED PRIVATE KEY-----`, depending on format.



NOTE: POM uploads support uploading certificate chains. All uploaded certificate chains must have an accompanying private key and must be ordered according to signing order. This means the Root certificate or the certificate closest to the root, must be placed first in the chain.

1. For **PKCS12** uploads:
    - Upload the `.p12` or `.pfx` file, which may one or more certificates and certificate chain/private key combinations.



1. *(Optional)* Provide a password if your certificate or key is encrypted.
   This applies to both **PKCS#8** and **PKCS#12** uploads.

2. Enter an **Alias** to identify the certificate and certificate chain/private key within the system. This only applies to **POM** uploads.

    - Certificate aliases can be edited on the Certificate details page, any other changes to the certificate would require a new entry. When an alias is updated, references to the certificate do not update and would need to be saved with the new alias.

Environment Variables | User Guide

> Source: https://guide.highbyte.com/configuration/administration/environment_variables/

Environment Variables

Environment variables are the global system environment variables configured on the host machine.

Important: Only environment variables that begin with `PUBLIC_` are available as references in the Intelligence Hub. This is to ensure that sensitive information is not inadvertently exposed.

These variables can be used for a more scalable approach to deployment. As an example, there may be multiple applications on a machine that reference a commonly used proxy. In this case, the proxy settings could be assigned to environment variables that can be referenced by all applications, and within the Intelligence Hub using an environment variable reference.

Enabling Environment Variables

Environment variables can be enabled in the UI on the Settings page under the Expressions section by toggling Allow Environment References.

Environment variables are loaded when the Intelligence Hub starts, or when the Allow Environment References setting is toggled.

Using Environment Variables

When enabled, environment variables are available via the Reference Panel under *Variables* | *Environment*. Like other inputs, `{{System.Environment.PUBLIC_VARIABLE}}` can be used to reference the value in supported fields in Instances, Connections, Inputs, Outputs, etc.

Similar to System Variables there are some fields (various Connection Settings for example) that do not support most inputs, but can be configured using environment variables.

Project | User Guide

> Source: https://guide.highbyte.com/configuration/administration/project/

Project

This section covers the various administrative project functions available in the Intelligence Hub.

- Export
- Import
- Project API

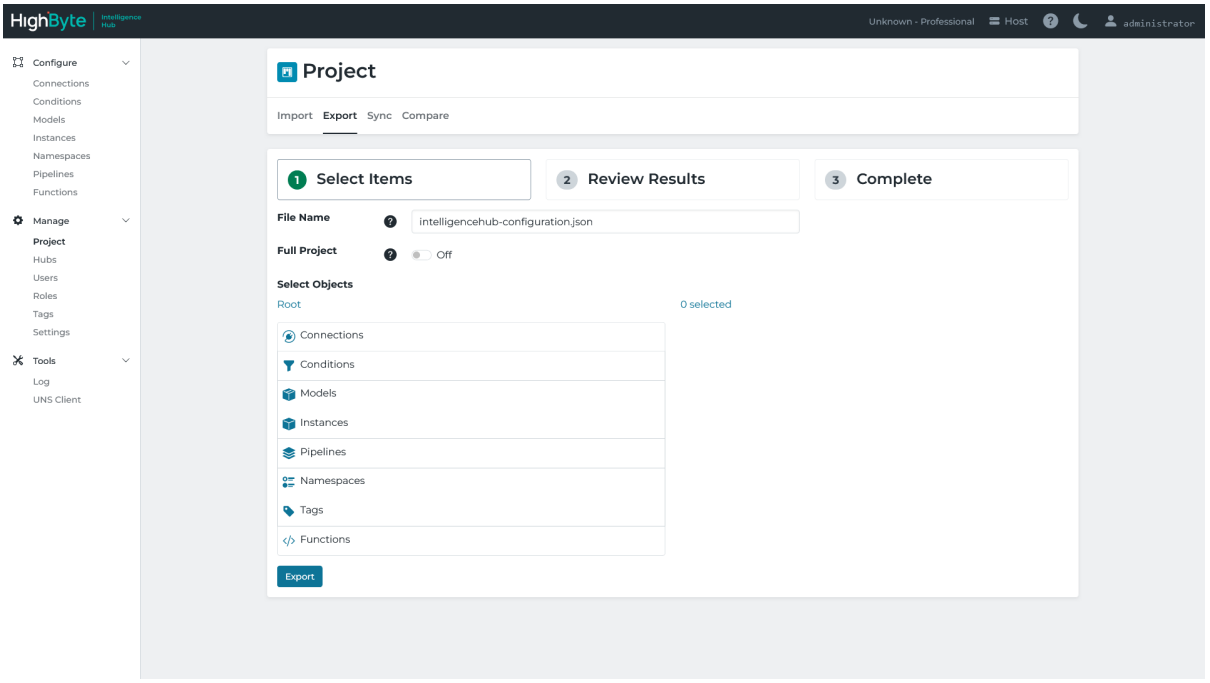
Export | User Guide

Source: https://guide.highbyte.com/configuration/administration/project/export/

Export

Project export can be used to quickly export a individual configuration objects or entire projects from the current hub. To export objects, perform the following steps.

1. Navigate to *Project* in the configuration's Main Menu and navigate to the *Export* tab.
2. To export the entire project, select *Full Project*. If *Full Project* is disabled, use the browse to select individual objects or tagged objects. When selecting a Tag, all objects with tagged with that Tag are selected. You can then de-select any objects that you don't wish to export.
3. Click *Export*.
4. The *Review Results* tab presents a JSON representation of the exported project.
5. To download a JSON file containing the project, click *Download*.



Import | User Guide

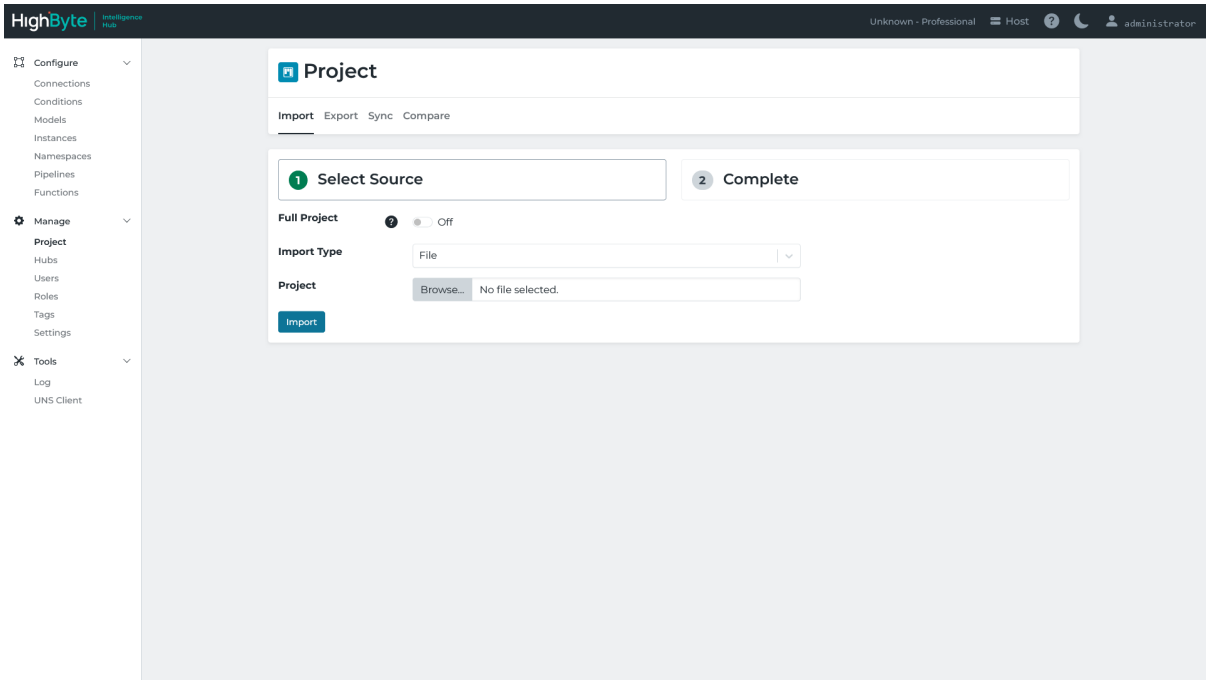
Source: https://guide.highbyte.com/configuration/administration/project/import/

Import

Project import can be used to quickly import individual configuration objects or entire projects to the current hub. To import objects, perform the following steps.

1. Navigate to *Project* in the configuration's Main Menu and navigate to the *Import* tab.
2. To replace the current project with the imported project, select *Full Project*. If *Full Project* is disabled, a partial import will occur and all incoming objects will be merged up into the current project.
3. The *Import Type* specifies the method used to provide the project. If *File* is selected, a JSON file containing the project must be specified. If *JSON* is selected, a JSON payload containing the project must be copied in.
4. Click *Import*.
5. The *Complete* tab shows whether project import succeeded for each object.

Project imports cannot be rolled back or reverted and partial success is possible if some objects fail to be imported.



Project API | User Guide

> Source: https://guide.highbyte.com/configuration/administration/project/project_api/

Project API

The Configuration API supports routes for importing/exporting full or partial projects. This can be useful in performing CI/CD deployments, backing up configuration, and performing mass configuration. The API endpoints are documented below.

Get Info

The get info route returns basic information about the Intelligence Hub like version, licensing, etc. It does not require authorization.

```
Method: GET
URL: http://localhost:45245/v1/info
Return: {"companyName":"HighByte","productName":"IntelligenceHub","productVersion":"4.0.0","
```

Authentication

Almost all of the API routes require authentication using a bearer token. To retrieve a bearer token, login using a username and password.

```
Method: POST
URL: http://localhost:45245/login
Body: {"username": "administrator", "password": ""}
Return: {"success": true, "message": null, "token": "Bearer 5af23835-ea58-43eb-9da2-41ef9bd3
```

Include the bearer token in API requests the require login as `Authorization: Bearer 5af23835-ea58-43eb-9da2-41ef9bd3e1d8`.

Project Export

Use this route to export the full project in JSON format.

```
Method: GET
URL: http://localhost:45245/v1/project/export
Return: Full JSON of the current configuration
```

Project Import

Use this route to import a full or partial project in JSON format.

```
Method: POST
URL: http://localhost:45245/v1/project/import
```

```
Body: A project in JSON format. Use the Project Export route or Project Export in the UI to
Return: Success/failure of each imported object in JSON format
```

Use this route to import a partial project in JSON format. This will merge the existing project with the new project, and update any objects with the same name.

```
Method: POST
URL: http://localhost:45245/v1/project/partial/import
Body: A full or partial project in JSON format. Use the Project Export route or Project Expo
Return: Success/failure of each imported object in JSON format
```

SAML and Active Directory | User Guide

Source: https://guide.highbyte.com/configuration/administration/saml_and_ad_setup/

SAML and Active Directory

External Identity Provider Setup

The Intelligence Hub supports using an External Identity Provider (IDP) for user login.

In this setup the hub defines the roles available to users, with appropriate restrictions. The IDP then assigns users to one or more hub roles by using the role names. After a user logs in through the IDP, the IDP sends the hub the list of roles the user is assigned to. The hub then matches the provided roles with the roles defined in the hub by name and restricts the users access.

Role names are matched using case-sensitive strings. Assume all settings described below are case-sensitive, and always match case with what is returned by the IDP.

Identity Provider Settings

IDPs are configured using the `intelligencehub-identityproviders.json` file located in application data directory. This file must be manually created before configuring or using any external IDP for user login.

The examples below cover how configure the `intelligencehub-identityproviders.json` to enable SAML and Active Directory IDPs.

Internal Authentication

Below is an example file that enables login using locally defined user accounts.

```
{
  "authentication": {
    "providers": ["internal"]
  }
}
```

SAML 2.0

Below is an example file that enables SAML and internal login. There are a number of SAML specific settings that are covered below.

The SAML response must return a custom attribute named 'roles' that contains the role names assigned to the user.

```
{
  "authentication": {
    "providers": ["saml2", "internal"],
    "saml2": {
      "sp.nameidformat":
      "urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress",
      "sp.entityid": "Intelligence Hub entity id",
      "sp.assertion_consumer_service.url":
      "http(s)://host:port/config/login/idp",
      "idp.entityid": "IDP entity id",
```

```
      "idp.single_sign_on_service.url": "IDP login in URL",
      "idp.single_sign_on_service.binding":
      "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect",
      "idp.x509cert": "IDPCertificate",
      "security.signature_algorithm":
      "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256",
      "security.digest_algorithm":
      "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256",
      "security.requested_authncontext":
      "urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport",
      "security.want_assertions_signed": false,
      "security.authnrequest_signed": false,
      "security.want_assertions_encrypted": false,
      "custom.roleNameAttribute" : "roles"
    }
  }
}
```

Settings	Description
`sp.nameidformat`	Name identifier used to represent the requested subject. Supported NameIdFormats: • `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress` • `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName` • `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName` • `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` • `urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos` • `urn:oasis:names:tc:SAML:2.0:nameid-format:entity` • `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` • `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` • `urn:oasis:names:tc:SAML:2.0:nameid-format:encrypted`
`sp.entityid`	Identifier of the Intelligence Hub entity.
`sp.assertion_consumer_service.url`	Where SAML assertions get sent to. Depending on where the Intelligence Hub is hosted.
`sp.privatekey`	The name of the private key used to sign SAML messages. The private key should also be stored under this name in the Intelligence Hub KeyStore. This field should only be used when the AuthnRequest is encrypted and signed. Note: Refer to the Certificate Section for more information.
`sp.x509cert`	The name of the public key used to encrypt the SAML response. The certificate should also be stored under this name in the Intelligence Hub KeyStore. This field should only be used when the AuthnRequest is encrypted and signed.

Settings	Description
	Note: Refer to the Certificate Section for more information.
`idp.entityid`	The identifier of the IdP entity.
`idp.single_sign_on_service.url`	Single sign on endpoint of the IdP. The URL where the AuthnRequest is sent.
`idp.single_sign_on_service.bin`	SAML protocol binding used to deliver the AuthnRequest message. Note: The Intelligence Hub currently only supports `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
`idp.x509cert`	Name of public x509 certificate of the IdP. The certificate should also be stored under this name in the Intelligence Hub KeyStore. Note: Refer to the Certificate Section for more information.
`security.signature_algorithm`	Algorithm that the toolkit will use on the signing process. Supported algorithms: • `http://www.w3.org/2000/09/xmldsig#rsa-sha1` • `http://www.w3.org/2000/09/xmldsig#dsa-sha1` • `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256` • `http://www.w3.org/2001/04/xmldsig-more#rsa-sha384` • `http://www.w3.org/2001/04/xmldsig-more#rsa-sha512`
`security.digest_algorithm`	Algorithm that the toolkit will use on the digest process. Supported algorithms: • `http://www.w3.org/2000/09/xmldsig#sha1` • `http://www.w3.org/2001/04/xmlenc#sha256` • `http://www.w3.org/2001/04/xmldsig-more#sha384` • `http://www.w3.org/2001/04/xmlenc#sha512`
`security.requested_authncontext`	Authentication context. If this field is not included no AuthContext will be sent in the AuthnRequest. Supported AuthContext: • `urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified` • `urn:oasis:names:tc:SAML:2.0:ac:classes:Password` • `urn:oasis:names:tc:SAML:2.0:ac:classes:X509` • `urn:oasis:names:tc:SAML:2.0:ac:classes:Smartcard` • `urn:oasis:names:tc:SAML:2.0:ac:classes:Kerberos`
`security.want_assertions_signed`	Set to true if the assertion received by the Intelligence Hub are signed. Set to false if it is not signed.
`security.authnrequest_signed`	Set to true if the AuthnRequest sent by the Intelligence Hub is signed. Set to false if it is not signed.
`security.want_assertions_encrypted`	Set to true if the assertions received by the Intelligence Hub are encrypted. Set to false if it is not.
`parsing.trim_name_ids`	Set to true if the name IDs received by the Intelligence Hub need to be trimmed. Set to false if it is not.

Settings	Description
`parsing.trim_attribute_values`	Set to true if the attribute values received by the Intelligence Hub need to be trimmed. Set to false if it is not.
`custom.roleNameAttribute`	The name of the attribute used to map incoming idp roles to Intelligence Hub roles. **Note**: *Defaults to* `roles`.

For more information on the settings, please see the following:

Active Directory

Below is an example file that enables Active Directory login. There are a number of AD specific settings that are covered below.

The AD response must include two key components. The attribute defined by the `userRoleAttribute` must define all the roles the user is assigned to. Each role is an object with many child attributes. The `roleNameAttribute` must define the child attribute that links to the hub role name. Both of these are required and are case sensitive.

```
{
  "authentication": {
    "providers": ["ad"],
    "ad": {
      "host": "localhost",
      "port": "389",
      "gatewayUsername": "User",
      "encryptGatewayPassword": "****",
      "domain": "domain.com",
      "ssl": false,
      "caAlias": "",
      "userFilter": "(&(objectClass=user)(!(objectClass=computer)))",
      "searchBase": "dc=domain,dc=com",
      "suffixGatewayUsername": true,
      "usernameAttribute": "sAMAccountName",
      "userRoleAttribute": "memberOf",
      "roleNameAttribute": "cn",
      "fullNameAttribute": "name",
      "phoneAttribute": "telephoneNumber",
      "emailAttribute": "mail",
      "smsAttribute": "mobile"
    }
  }
}
```

Settings	Description
`host`	Hostname of the server hosting AD.
`port`	Port of the server hosting AD.
`gatewayUsername`	User used to perform user lookups in AD.
`encryptGatewayPassword`	Password of the user used to perform user lookups in AD. When manually creating the file, specify the password in plain text. When the Intelligence Hub first runs it will encrypt the password and write it back out to the file.
`domain`	Domain of AD.

Settings	Description
`ssl`	If SSL is enabled when connecting to the AD server.
`caAlias`	Name of public x509 certificate of the AD server. The certificate should also be stored under this name in the Intelligence Hub KeyStore. Note: Refer to the Certificate Section for more information.
`userFilter`	Applied list filter when performing user lookups in AD. **Note**: *Defaults to* `(&(objectClass=user)(!(objectClass=computer)))`.
`searchBase`	Applied search filter when performing user lookups in AD. **Note**: *Defaults to the domain name.*
`suffixGatewayUsername`	Set to true to append `@domainName` to the gateway username
`usernameAttribute`	Attribute used to retrieve usernames. This is the username we compare against the user attempting to login. **Note**: *Defaults to sAMAccountName.*
`userRoleAttribute`	Attribute used to retrieve user roles. **Note**: *Defaults to memberOf.*
`roleNameAttribute`	Attribute to map Intelligence Hub user roles to the returned user roles (userRoleAttribute). **Note**: *Defaults to* `cn`.
`fullNameAttribute`	Attribute used to get the user's full name. **Note**: *Defaults to* `name`.
`phoneAttribute`	Attribute used to get the user phone number. **Note**: *Defaults to* `telephoneNumber`.
`emailAttribute`	Attribute used to get the user email. **Note**: *Defaults to* `mail`.
`smsAttribute`	Attribute used to get the user sms number. **Note**: *Defaults to* `mobile`.

Troubleshooting

Errors with IDP authentication are logged to the event log for troubleshooting. For advanced debugging, enable the Debug Logging.

If the user does authenticate with the IDP and we fail to match any of the IDP roles with hub roles, the user will remain logged in but will have no access rights, and will see permission errors in the UI.

SAML2 Advanced Logging

SAML2 supports advanced debugging by enabling logging. To do this, create a logback.xml in the appData directory. Place the following in the file.

```
<configuration scan="true" scanPeriod="10 seconds">
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>out/runtime.debug.log</file>
        <append>false</append>
        <immediateFlush>true</immediateFlush>
        <encoder>
            <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="OFF">
        <appender-ref ref="FILE" />
    </root>

    <logger name="com.onelogin.saml2.authn" level="ALL">
        <appender-ref ref="FILE" />
    </logger>

</configuration>
```

Stop and restart the runtime with the following command added to the command line.

```
-Dlogback.configurationFile=logback.xml
```

The command can be added to the start script for the operating system.

```
// Example adding to start-windows.bat
java -Dlogback.configurationFile=logback.xml -cp "intelligencehub-runtime.jar;lib/*" com.hig
```

Detailed SAML2 debug logs will be logged to a runtime/out/runtime.debug.log file.

Secrets | User Guide

Source: https://guide.highbyte.com/configuration/administration/secrets/

Secrets

Secrets include connection passwords, tokens, and other sensitive information the Intelligence Hub requires to connect to external systems. This section covers the different ways to manage secrets in the Intelligence Hub.

Encryption/Decryption

In all cases, secrets are stored securely on the local file system using a private key to encrypt and decrypt the secrets. The key is located in the `intelligencehub-certificatestore.pkcs12` file. The key is generated when first running the Intelligence Hub, and re-created any time the file or key cannot be loaded.

Inline Secrets

By default, secrets entered for a Connection (ex. a password) are encrypted and saved in the `intelligencehub-configuration.json` along with all project configuration. Below is an example of an encrypted field for a connection.

```
{
  "password" : {
    "type" : "Encrypted",
    "value" : {
      "keyId" : "/68PfkTLjYIbwiCzbi8++A==",
      "iv" : "ohKWcdoU+99Ofjt9k8in/g==",
      "ciphertext" : "nzU9pFgwUK5DXrh/Pgyz4w=="
    }
  }
}
```

Inline secrets are convenient and simple but can become problematic when managing the same project across different environments.

External Secrets

External Secrets may be a better solution in cases where a project is moved between environments that have different credentials. An example of this could be Dev, QA, and Production. With External Secrets the `intelligencehub-configuration.json` references a secret by name, but the secret is stored and encrypted in a `intelligencehub-secrets.json`. This way the project file can remain the same between environments and only the secrets file changes.

Secrets can be managed in the Settings | Secrets tab, and then referenced in the Connection settings for each Connection. Below is an example of the secret reference in the `intelligencehub-configuration.json` file.

```
{
  "password" : {
    "type" : "Reference",
    "value" : "mypassword"
  }
```

```
}
```

Sharing Projects with Secrets

In order for an Intelligence Hub to load secrets, either inline or external, it must have the same key as the Hub that encrypted the secrets. This means it must share the `intelligencehub-certificatestore.pkcs12` file. If the Hub loading the secrets does not have the same key, the secrets will fail to load and must be manually re-entered to be encrypted by the local key.

Project Import/Export

Project import can import secrets in plain text, encrypted, or referenced formats. Although plain text secrets are not recommended, the import will encrypt the secret using the local key before storing it in the configuration file. If the secret is imported as encrypted, the local key is used to decrypt the secret. If this fails, the user is warned and they must re-enter the secret before using the connection. If the imported secret is a reference, the local `intelligencehub-secrets.json` must contain the reference, otherwise a runtime error occurs when using the connection.

Project export can export secrets as inlined or referenced. Project export never exports secrets in plain text.

System Variables | User Guide

> Source: https://guide.highbyte.com/configuration/administration/system_variables/

System Variables

System variables are a way to share projects across multiple installations of the Intelligence Hub but modify some settings for a given deployment. As an example, all deployments might publish to the same MQTT broker but each must publish on a topic specific to the site where the Intelligence Hub is deployed.

System Variables can be configured in the UI under Settings | Variables.

System Variables can also be loaded from the `intelligencehub-systemvariables.json` file when the Intelligence Hub starts. The `intelligencehub-systemvariables.json.template` provided in the install provides an example of the format.

```
[
    {
        "name": "variable_name",
        "value": "variable_value"
    }
]
```

The file contains variable names and values. Once loaded, these variables are available via the Reference Panel under *Variables* | *System*. Like other Inputs, `{{System.Variables.SiteID}}` can be used to reference the value in supported fields in Instances, Connections, Inputs, Outputs, etc. For example, a system variable can control the IP address of the OPC UA server, and this can be changed for Production versus Development environments.

Similar to Environment Variables there are some fields (various Connection Settings for example) that do not support most inputs, but can be configured using system variables.

The variables can be viewed and set through the UI on the Settings | Variables page. Any changes are persisted to the file and take effect immediately. The Hub view in Central Configuration mode also provides a way to see and change variables on remote hubs.

Users and Roles | User Guide

Source: https://guide.highbyte.com/configuration/administration/users_and_roles/

Users and Roles

Users can be created and assigned roles and/or claims to allow them to perform certain types of operations.

To create a new User

1. Navigate to *Users* in the configuration's Main Menu. Here you can add, edit, and view users. To add a user, click the **New User** button.
2. Enter the *Username* and *Password*. *Click Next* to continue.
3. Select the *Roles* and/or set the *Claims* the user inherits. Click *Submit* to add the user. Claims are specified through the usage of resources and actions. Resources are the object (e.g., connection) and actions are the operation that can be performed in CRUD terminology (create, read, update, delete) on a resource.

To create a new Role

Roles can be created and assigned claims that are inherited by one or more users. The roles define the type of operations a class of users may perform.

1. Navigate to *Roles* in the configuration's Main Menu. Here you can add, edit, and view roles. To add a role, click the **New Role** button.
2. Enter a *Name* to represent the new role.
3. Set the *Claims* the user assigned to this role should inherit. Click *Submit* to add the role.

Claim Information

Supported Resources

Resource	Description
*	Specifies all resources
broker	MQTT Broker
certificate	Certificates
connection	Connections, includes inputs and outputs
condition	Conditions
function	Global functions
instance	Instances
log	The event log
model	Models
pipeline	Pipelines
network	Network hubs, groups, and sync operations
Settings	System level settings

Resource	Description
tag	Tags
user	Users and roles

Supported Actions

Action	Description
*	Specifies all actions
create	Create new resource
read	Read resource
list	Retrieve a list of resources
update	Edit resource
delete	Delete resource
execute read	Read data for a source (ex. Input or Instance)
execute write	Write data to a target (ex. Output or Pipeline)
publish	Publish to an MQTT broker topic
subscribe	Subscribe to an MQTT broker topic

Supported Scopes

Scope	Description
Contains Tag	Limits access to resources tagged by one of the specified tags (this scope only applies to Connections, Conditions, Models, and Instances)

API Keys

API Keys can be created and assigned to a user to limit access to the REST Data Server.

User API keys have a name, expiration date (optional) and description. Once created, the API key can be used as part of the REST Data Server login. See Application Settings for details on token based login.

As an example, to limit write access to an OPC UA connection through the REST Data Server, create a claim for the user with the `execute write` action on a `connection` resource,
tag the connection with a new tag (ex. `opcua_write`), and include the tag in claim.

Condition | User Guide

Source: https://guide.highbyte.com/configuration/condition/

Condition

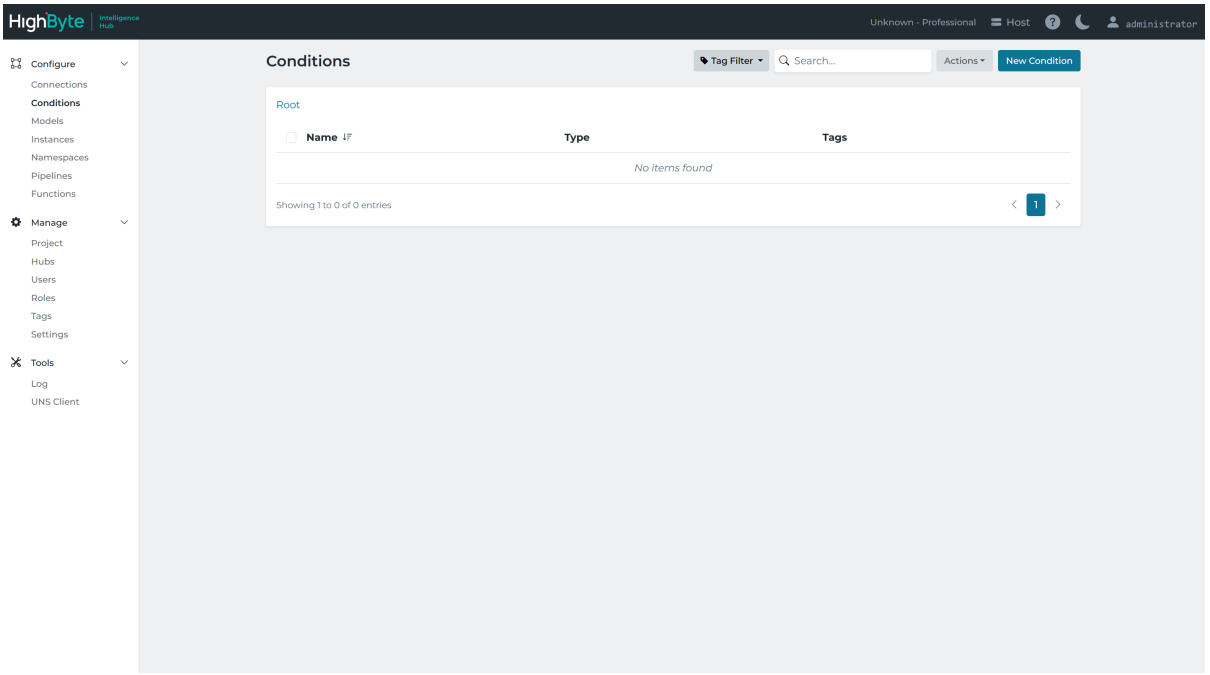
Conditions are a way to manipulate raw data read from an input. There are several types of conditions each with applicable settings. One example of a Condition is the ability to apply deadband to a raw input value, only providing data changes if the value changed by a set threshold.

The Intelligence Hub supports four types of condition:

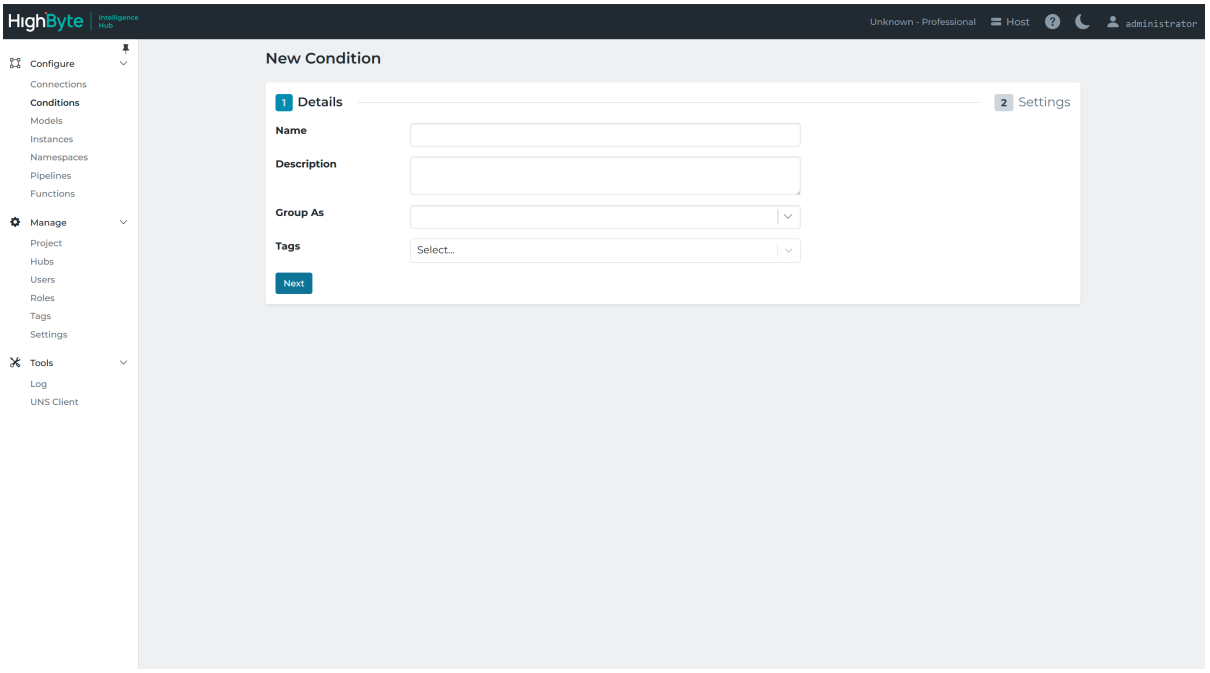
- Aggregate
- Alarm
- Custom
- Deadband

Create a Condition

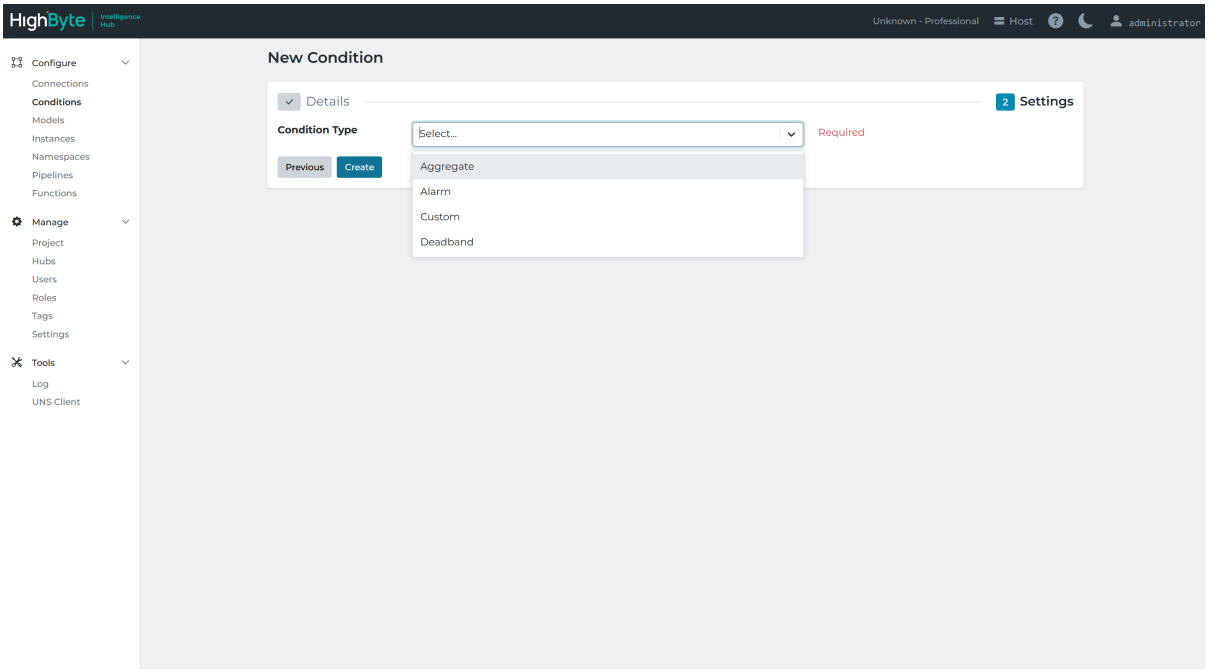
1. Navigate to *Conditions* in the configuration's Main Menu. Here you can add, edit, and view conditions. To add a condition, click the **New Condition** button.



2. Enter a *Name* to represent the condition. Names can only contain alphanumeric and underscore characters (e.g., A-Z, a-z, 0-9 or _). Optionally enter in a *Description* for the condition. Click *Next* to continue.



3. Select the *Type*.



4. Set the Conditions *General* and *Type Specific Settings*. Click *Submit* to add the condition and return to the main condition page. General and Type Specific Settings are described below.

General Settings

Attribute	Description
Sources	Conditions reference one or more sources of data. Sources can be an Input and/or Instance. Conditions read the source data and provide specific processing of the data before exposing the result inputs into other configuration elements.

Type Specific Settings

- Aggregate
- Alarm
- Custom

- Deadband

Aggregate | User Guide

> Source: https://guide.highbyte.com/configuration/condition/aggregate/

Aggregate

The aggregate condition exposes an input for each aggregate computed on the source. As an example, if `{{Connection.opc.tag}}` is an input, the aggregate condition exposes `{{Condition.MyAggregate.opc_tag_<aggregate>}}` as an input. The aggregate condition only stores data changes, and aggregates are computed at the time of the read.

Numerical aggregates (AVG, MIN, MAX, DELTA) are not supported on non-numeric types like strings or arrays. If the aggregate source is a complex object, the numerical aggregate is applied to each numerical value in the complex object.

Settings

Scan Interval

The rate at which the sources are scanned and stored.

Storage Interval

How long data is retained. Data older than the storage interval is discarded. It's common to set the storage interval to longer than the window interval. This way the window interval can expand, and still be valid if the stored data is still available. As an example, you may want to store data for two days, and change the window to get the count of yesterday's shift totals. The storage interval should never be less than the window interval. In this case the window will always be missing data.

Type

The type of windowing the aggregate supports. See window types below.

Enabled

When disabled, the aggregate no longer polls the sources for new data. Existing data remains in the aggregate storage and can be read.

Window Types

Sliding

Sliding window includes data from the time of the read, minus the interval time. As an example, a sliding window with a window interval of 10 minutes calculates an average for the last 10 minutes of data. Data older than 10 minutes is ignored. The Window Interval controls the length of the sliding window.

Fixed

Fixed window sets the explicit start and stop time of the window. Start time should always be older than the stop time, and they can be fixed times or dynamic references to other inputs. Fixed window calculates the aggregate using data between the start and stop times.

- **Start Window**: The absolute time of the start of the window. Accepts a static UTC ISO8601 (ex. `2022-03-29T20:59:45Z`) or epoch in milliseconds (ex. `1648587633385`) format. Also accepts a reference to an input that provides the time format. The start window should be older than the stop window

- **Stop Window**: The absolute time of the end of the window. Accepts the same as the Start Window, but the time should be newer. If the stop time is in the future, the stop time is set to the current time of the read until the stop time is reached.

Supported Aggregates

RAW

Returns an array of the raw data collected for the source over the window. The data includes a time and value attribute, allowing for custom expressions/calculations in instances via the expression engine. If there is no data for the window, an empty array is returned. Values can be complex or simple data types.

COUNT

Returns the number of values for the source in the time window. Only data changes are included. If there are no data changes for the window the count returns 0.

AVG

Returns the time weighted average of the value over the time window. This includes the value at the start of the window and the value at the end. The calculation is as follows. `AVG = (V1 * Tv1/Ttotal) + (V2 * Tv2/Ttotal) ...`

Note that `V1` is the value at the start of the window. `Tv1` is how long the value was V1. `Ttotal` is the total window length. If there is no data for the window, the read returns good quality no data.

MIN

Returns the minimum value over the time window.

Note this includes the value at the start of the window, so if the value changed but was never lower than the value at the start of the window, the value at the start is reported. |

MAX

If there is no data for the window, the read returns good quality no data. Returns the maximum value over the time window. Like MIN, this includes the value at the start of the window. If there is no data, the read returns good quality no data.

DELTA

Returns the delta of the most recent value versus the oldest value in the window. This includes the value at the start of the window. If there are no data changes for the window, DELTA returns 0.

If there is no data for the window, the read returns good quality no data.

Alarm | User Guide

> Source: https://guide.highbyte.com/configuration/condition/alarm/

Alarm

The Alarm Condition is used to monitor data sources and alarm when they stop providing data. For example, if you expect to receive data from a sensor via MQTT every 5 minutes, use the Alarm Condition to generate an alarm when the sensor hasn't sent data for 10 minutes.

The condition exposes an input for each source, and an "all" input that is an aggregate alarm for all configured sources. The condition inputs return an isAlarm flag to indicate an alarm state. When the source is in alarm the input also returns a default error message and the name of the source and interval.

It's best to use the condition in a separate pipeline that triggers off the isAlarm flag and then sends either the default or a custom error message out to a source system (ex. Splunk, SQL, MQTT, etc).

Settings	Description
Alarm Type	The type of alarm. Currently only Time is supported.
Alarm Interval	How long the source needs to be either in non-Good quality or not provide an updated value of Good quality before the source goes into an alarm state. For example, if the interval is 1 minute and the source is always bad quality (ex. not connected) the source will alarm after 1 minute.

Custom | User Guide

Source: https://guide.highbyte.com/configuration/condition/custom/

Custom

Custom Conditions are designed to prepare raw input data for modeling. Some examples of this are converting XML to JSON, converting raw binary streams to data types, process counters, and more.

Custom Conditions expose an input for each source. If the source is `{{Connection.opc.tag}}` the conditioned input is `{{Condition.myCondition.opc_tag}}`. Reading this value returns the value of the input run through the expression.

Custom Conditions maintain the state of the last read and last returned values per source, allowing flexibility in handling things like state changes.

Expression

The JavaScript expression that runs every time a source is read. The expression contains the following variables that are kept for each source in the condition. These values are persisted on disk and persist between condition reads.

Syntax	Description
`{{this.currentValue}}`	The current value of the source being read. For example, this would be the current value of the OPC tag or MQTT topic.
`{{this.lastValue}}`	The previous value for the source that was read the last time the expression was executed. For example, if the currentValue on the first read is 100, the lastValue on the next read is 100. This value is null when reading a source for the first time. This value is only cached if `{{this.lastValue}}` is used in the expression.
`{{this.lastReturnValue}}`	The last value returned by the expression for the source. This is null on the first read of the source. This value is only cached if `{{this.lastReturnValue}}` is used in the expression.

In addition to the above variables, other sources can be directly included in the expression by dragging/dropping the reference from the reference panel.

Custom Condition Examples

Below are common examples of Expressions. Note Expressions use JavaScript (ECMAScript/ECMA-262 specification).

Convert Named Object to Array

Assume the input payload looks like this.

```
{
  "tag1":{
    "value": 100,
    "timestamp": "2023-05-31T12:30:56Z",
    "quality": "good"
  },
  "tag2": {
    "value": 60,
    "timestamp": "2023-05-31T12:30:56Z",
```

```
    "quality": "good"
  }
}
```

We want to convert the payload to this.

```
[
  {
    "name": "tag1",
    "value": 100,
    "timestamp": "2023-05-31T12:30:56Z",
    "quality": "good"
  },
  {
    "name": "tag2",
    "value": 60,
    "timestamp": "2023-05-31T12:30:56Z",
    "quality": "good"
  }
]
```

The following expression for the condition accomplishes this.

```
var rawObject = {{this.currentValue}}
var arrayOut = []

// Iterate each tag in the payload
for (const tagName in rawObject){
    // Create new object and add name
    let tagObject = {...rawObject[tagName]}
    tagObject["name"] = tagName

    // Add to the array
    arrayOut.push(tagObject)
}

// Return the array
arrayOut
```

Convert Array to Named Objects

Assume we want the reverse of the above example. In this case the raw data comes in as an array, and we want to convert it to named objects.

```
var rawArray = {{this.currentValue}}
var objectOut = {}

// Iterate each object in the array
for (const tag of rawArray){

    // Get the tag name
    let tagName = tag["name"]

    // Add the tag to the object, using the name as the key
```

```
    objectOut[tagName] = tag

    // Delete the name field
    delete objectOut[tagName]["name"]
}

// Return the object
objectOut
```

Return New Rows Only

In cases where an input returns all data, and you're only concerned about new data, a condition can be used to filter out data that has already been read. Ideally this would be done by the input, for example using indexing in SQL, but in cases where it can't a condition can perform the filtering.

Assume the raw input returns the following data.

```
[
  {
    "id": 1,
    "name": "tag1",
    "value": 100,
    "timestamp": "2023-05-31T12:30:56Z",
    "quality": "good"
  },
  {
    "id": 2,
    "name": "tag2",
    "value": 60,
    "timestamp": "2023-05-31T12:30:56Z",
    "quality": "good"
  }
]
```

On the first read this condition returns the two rows. On the second read the condition returns an empty array. If a new row is added with a new id, the third read returns the new row.

```
var currentReadRows = {{this.currentValue}}
var lastReadRows = {{this.lastValue}}
var retRows = []

if (lastReadRows == null){
    // First read, there is no last read
    currentReadRows
}
else{
    // Iterate each row from the latest read
    for (const currentRow of currentReadRows){
        let foundRow = false;

        // Iterate each row from the last read and see if there is a match by the id propert
        for (const lastRow of lastReadRows){
            if (lastRow._id == currentRow._id){
                foundRow = true
                break;
            }
```

```
        }

        // If no match is found, return the new row
        if (!foundRow)
            retRows.push(currentRow)
    }

    retRows
}
```

Deadband | User Guide

Source: https://guide.highbyte.com/configuration/condition/deadband/

Deadband

The deadband condition exposes an input for each source. If the source is `{{Connection.opc.tag}}` the deadband conditioned input is `{{Condition.myDeadband.opc_tag}}`. Reading this value returns the deadbanded opc tag value.

Deadband works on simple numeric data types and numeric types in complex data. For complex data, each numeric attribute is deadbanded separately.

Settings	Description
Absolute Deadband	The absolute value that the source must change to reflect the new value in the output. The source must change more than this absolute value. As an example, if the setting is 5 and the source changes from 0 to 4, the deadbanded value remains 0. If the value changes to 5, it remains 0. If the value changes to 6, the new deadband value is 6.

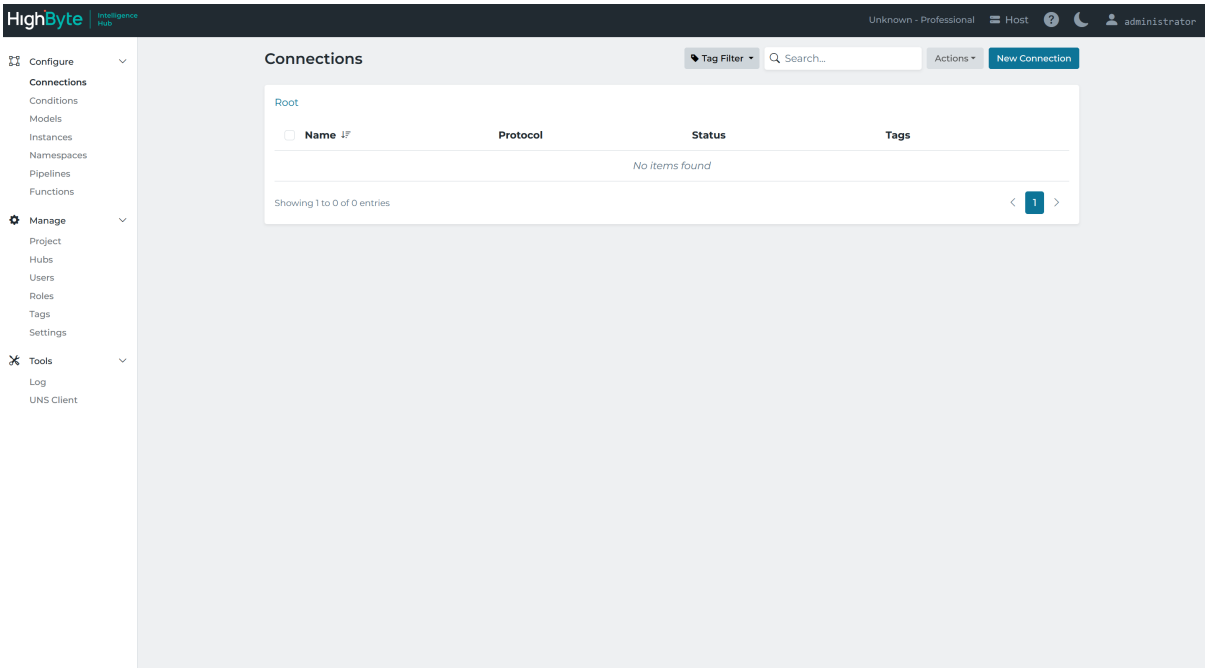
Connect | User Guide

> Source: https://guide.highbyte.com/configuration/connect/

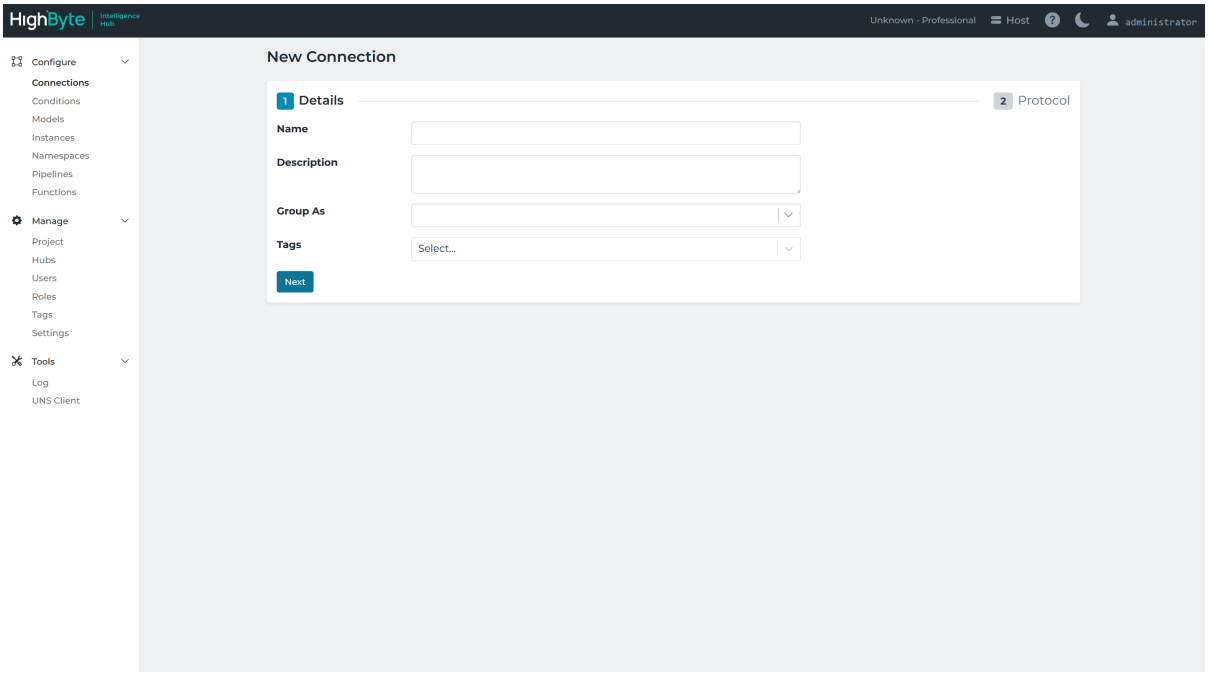
Connect

A Connection represents a path to a data source that contains inputs that can be read and/or outputs that can be written to. To view all Connections, click *Connect* in the configuration's Main Menu. To create a new connection, do the following.

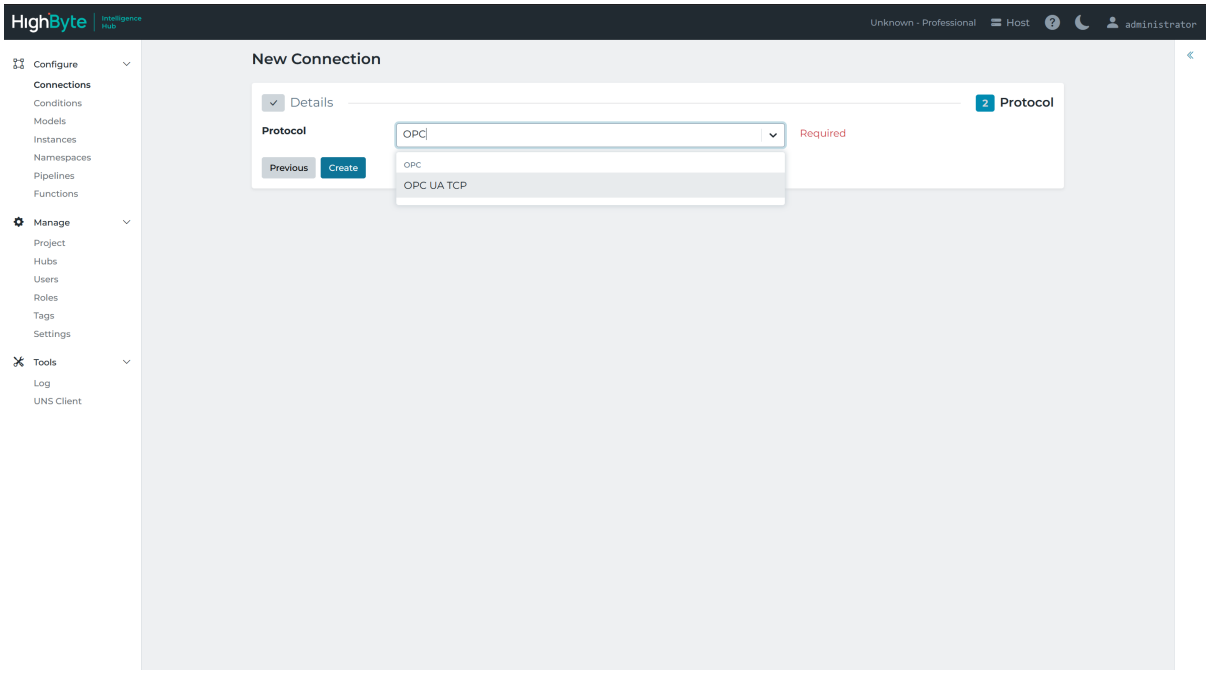
1. Click *Create Connection*.



2. Enter a *Name* to represent the connection. Names can only contain alphanumeric and underscore characters (e.g., A-Z, a-z, 0-9 or _). Optionally enter in a *Description* for the connection. Click *Next* to continue.



3. Select the *Protocol*. Depending on the protocol. Click *Next* to continue.



4. Set any Connection *General* and *Protocol Specific Settings*. Click *Submit* to add the connection and return to the main Connection page. General Settings are described below.

General Settings

The following settings are supported by all Connections:

Store and Forward

All writes are in first in first out order (FIFO). When store and forward is enabled, all writes are first sent to disk and only cleared from disk after the write succeeds. Writes are stored in a sqlite file located in the store and forward directory configured in the application settings. The file is named `intelligencehub-writestore-{{connection name}}.db`, and contains a table for each unique output for the connection. When store and forward is disabled, failed writes are not retired.

To view the status of store and forward, and manually clear stored writes, see the Connection Statistics section below.

Setting	Description
Enabled	Specifies whether store and forward is enabled for the connection.
Max Entries Per Output	Maximum number of writes per output to store. Once the maximum is reached, any new writes are discarded. Upon re-establishing a connection, stored outputs are written in the same order they were entered into the store (FIFO). An output's write entry is only removed from the store once it has been successfully written to the destination. All stored output writes are written as fast as possible to ensure the delivery of newer data is not stalled.
Failure Wait Interval	How long to wait before retrying to write the next batch of stored entries when a connection has failed. The interval can be specified in milliseconds, seconds, minutes, hours, or days. Valid range is between 10 milliseconds and 36 days. Once the connection is re-established, writes are handled as described above.

Write

Setting	Description
Flatten Modeled Values	When enabled, hierarchy in the output data is flattened. For example, output data that contains `parent { child: 123 }` is flattened to parent_child: 123. This option is not shown for connections that do not support hierarchy (e.g., SQL) and values are flattened by default.

Statistics

The Connection Statistics tab shows the overall health of the connection and the status of writes.

Connection Statistics

Statistic	Description
Healthy	Shows if the connection is established or is in error.
Error	If the connection is in error, this shows the last error message.
Pending Writes	The number of writes that are pending in memory or in the store and forward file. For healthy connections this should not be growing over time.
Pending Write Size	The amount of memory the has been allocated for pending writes.
Completed Writes	The number of writes that have been completed by the connection across all outputs since the last time the connection was saved or reset.
Writes/Second	The number of writes per second being completed by the connection across all outputs. This is an instantaneous rate based on the writes completed in the last 5 seconds.
Dropped Writes	The number of writes that have been dropped by the connection because the in memory buffer of 10000 writes per output OR the store and forward size has been exceeded.

Output Statistics

Clear Selected
Delete all of the pending writes for the selected outputs

Reset
Reset the store and forward database for this connection. This will delete all pending writes for all outputs on this connection, and clean up the database's allocated space.

Statistic	Description
Pending Writes	The number of writes that are pending in memory or in the store and forward for the output.
Completed Writes	The number of writes that have been completed for the output since the last time the connection was saved or reset.
Writes/Second	The number of writes per second being completed by for the output. This is an instantaneous rate based on the writes completed in the last 5 seconds.
Dropped Writes	The number of writes that have been dropped for the output because the in memory buffer of 10000 writes per output OR the store and forward size has been exceeded.

Protocol Specific Settings

See Connections for Protocol Specific Settings.

Connections | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/

Connections

HighByte Intelligence Hub supports many types of connections for retrieving and sending data. Below is a comprehensive list linking to protocol specific settings.

- AI
  - Amazon Bedrock
  - OpenAI
  - Azure OpenAI
  - Google Gemini
- Amazon
  - Redshift
  - S3
  - S3 Tables
  - IoT SiteWise
  - Data Firehose
  - Kinesis Data Streams
- Microsoft Azure
  - Blob Storage
  - OneLake
  - Event Hubs
  - IoT Edge
  - IoT Hub
  - Microsoft Fabric
- Google
  - BigQuery
  - PubSub
- Historians
  - AspenTech IP.21
  - InfluxDB
  - PI System
  - TimescaleDB
- SQL
  - JDBC Driver
  - Microsoft SQL Server
  - MySQL
  - Oracle Database
  - PostgreSQL
  - SQLite
- Files
  - CSV
  - File
  - Parquet
- Databricks Client
- Databricks SQL
- Ignition Module
  - Installing Module
- Kafka
- Modbus TCP
- MQTT

- OPC UA TCP
  - Troubleshooting
  - OPC-UA CSV Import
- MCP Client
- REST Client
- Snowflake Streaming
- Snowflake SQL
- Sparkplug
- Webhook

AI | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/ai/

AI

- Amazon Bedrock
- OpenAI
- Azure OpenAI
- Google Gemini

Amazon Bedrock | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/ai/amazon_bedrock/

Amazon Bedrock

The Amazon Bedrock connection allows the Intelligence Hub to interact with AWS Bedrock for generative AI and LLM workflows. This connection enables sending prompts and receiving responses from models hosted on Bedrock. Below are details on the connection and input settings.

Use care when integrating this connection into production workflows. Due to the rapid pace of AI and LLM development, providers have made breaking changes to their APIs in the past. While we aim to respond quickly to such changes, they may temporarily disrupt production systems.

Connection Settings

Setting	Description
Access Key	The access key used to authenticate with the Bedrock service.
Secret Key	The secret key used to authenticate with the Bedrock service.
Region	The AWS region where the Bedrock service is hosted.
Endpoint	(Optional) Base URL for the Bedrock API. If not provided, the default endpoint is used.

Input Settings

Setting	Description
Model Id	The model ID provided by AWS when creating a model in Bedrock.
Instructions	(Optional) The prompt or instructions to guide the model's. response.
Message	The message to be sent to the model. This is the input the model will process based on the instructions.
Response Format	The format in which the model should respond. When set to JSON the connection will attempt to parse the response as JSON. If the response is not valid JSON the read fails.

Azure OpenAI | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/ai/azure_openai/

Azure OpenAI

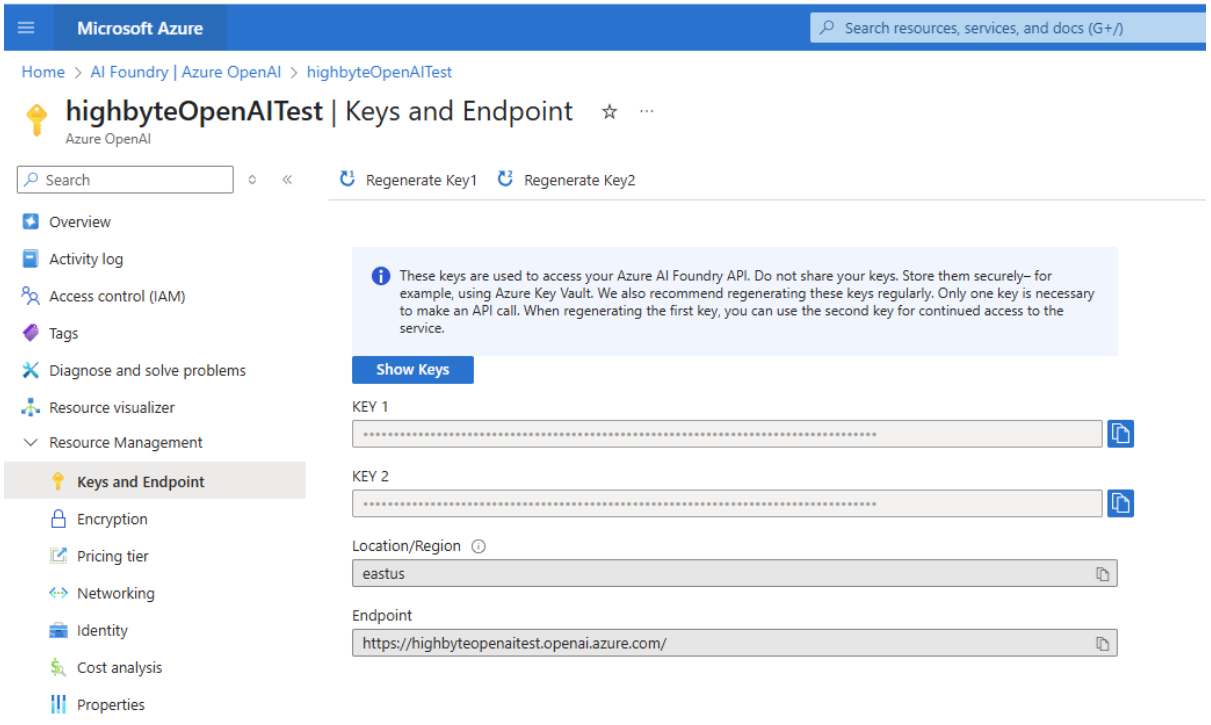
The Azure OpenAI connection allows the Intelligence Hub to interact with Azure-hosted OpenAI models for generative AI and LLM workflows. This connection enables sending prompts and receiving responses from models deployed in Azure. Below are details on the connection and input settings.

Use care when integrating this connection into production workflows. Due to the rapid pace of AI and LLM development, providers have made breaking changes to their APIs in the past. While we aim to respond quickly to such changes, they may temporarily disrupt production systems.

Connection Settings

Setting	Description
Key	The API key used to authenticate with the Azure OpenAI service.
Endpoint	The URL for the Azure OpenAI API endpoint.

Below is an example screenshot of where to find these settings in Azure.



Input Settings

Setting	Description
Deployment	The deployment name or ID for the model to use. This is typically a deployment name provided by Azure OpenAI.

Setting	Description
Instructions	(Optional) The prompt or instructions to guide the model's response.
Message	The message to be sent to the model. This is the input the model will process based on the instructions.
Response Format	The format in which the model should respond. When set to JSON the connection will attempt to parse the response as JSON. If the response is not valid JSON the read fails.

Google Gemini | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/ai/gemini/

Google Gemini

The Google Gemini connection allows the Intelligence Hub to interact with Gemini models for generative AI and LLM workflows. This connection enables sending prompts and receiving responses from models hosted by Google Gemini. Below are details on the connection and input settings.

Use care when integrating this connection into production workflows. Due to the rapid pace of AI and LLM development, providers have made breaking changes to their APIs in the past. While we aim to respond quickly to such changes, they may temporarily disrupt production systems.

Connection Settings

Setting	Description
API Key	The API key used to authenticate with the Google Gemini service.

Input Settings

Setting	Description
Model	An identifier for the model to use. This is typically a model name or ID provided by Google Gemini. A common model is “gemini-2.0-flash” for example.
Instructions	(Optional) The prompt or instructions to guide the model’s response.
Message	The message to be sent to the model. This is the input the model will process based on the instructions.
Response Format	The format in which the model should respond. When set to JSON the connection will attempt to parse the response as JSON. If the response is not valid JSON the read fails.

OpenAI | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/ai/openai/

OpenAI

The OpenAI connection allows the Intelligence Hub to interact with OpenAI models for generative AI and LLM workflows. This connection enables sending prompts and receiving responses from models hosted by OpenAI. Below are details on the connection and input settings.

Use care when integrating this connection into production workflows. Due to the rapid pace of AI and LLM development, providers have made breaking changes to their APIs in the past. While we aim to respond quickly to such changes, they may temporarily disrupt production systems.

Connection Settings

Setting	Description
Password	The password used to authenticate with the OpenAI service. This is typically an API key.
Base URL	(Optional) Base URL for the OpenAI API. If not provided, the default endpoint is used. Many LLM engines support the OpenAI API, and this can be used to interact with local LLM endpoints (ex. LM Studio).

Input Settings

Setting	Description
Model	An identifier for the model to use. This is typically a model name or ID provided by OpenAI. Common models are “gpt-4” or “o4-mini” for example.
Instructions	(Optional) The prompt or instructions to guide the model’s response.
Message	The message to be sent to the model. This is the input the model will process based on the instructions.
Response Format	The format in which the model should respond. When set to JSON the connection will attempt to parse the response as JSON. If the response is not valid JSON the read fails.

Amazon | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/

Amazon

- Redshift
- S3
- S3 Tables
- IoT SiteWise
- Data Firehose
- Kinesis Data Streams

Redshift | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/aws/amazon_redshift/

Redshift

Connection Settings

JDBC Connection String

The connection string provided by AWS via the Redshift configuration.
Check the Redshift configuration in AWS to ensure external connections are enabled.

Username

Username for authentication.

Password

Password for authentication.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input/Output Settings

See JDBC Driver Settings for input and output settings.

S3 | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/amazon_s3/

S3

Connection Settings

Access Key

IAM created user access key with S3 permission.

Secret Key

IAM provided secret key.

Region

Region of the Amazon Kinesis Data Streams instance (e.g., `us-east-1`)

Endpoint Override

The URL for a custom S3-compatible service. Leave blank to use the default AWS S3 endpoint for the selected region. This can also be set using the `AWS_ENDPOINT_URL_S3` environment variable.

Proxy Endpoint

The URL for a custom S3 proxy. Leave blank to route traffic directly to AWS S3.

Input Settings

Type

Specifies the type of input operation. *List* will return the a set of directories and files at a specified path. *Read* will retrieve the data associated with a specific file.

List

Bucket Name

The name of the bucket to read data from.

Key

The name of the file to read.

Recursive

Specifies the set of files to list. When *false*, only immediate children of the directory are returned. When *true*, all files and directories under this path are included recursively.

Time Filter

Enables filtering files based on an ISO-8601 time. When enabled only returns files updated after the provided time.

Read

Bucket Name

The name of the bucket to read data from.

Key

The name of the file to read.

Note: *Retrieving files stored in the S3 Glacier Flexible Retrieval storage class, the S3 Glacier Deep Archive storage class, the S3 Intelligent-Tiering Archive Access tier, or the S3 Intelligent-Tiering Deep Archive Access tier is currently not supported.*

Encoding

The character set used to decode the contents of the file. Select **Binary** to read the file as binary and skip decoding. Select **Auto** to decode the file using the BOM of the file. Otherwise, use a specific encoding type to decode the file.

Include Metadata

Includes metadata specific to S3. Will include System-defined and User-defined object metadata.

Output Settings

Bucket Name

The name of the bucket to write output data to. This field supports dynamic output references.

Key

Specifies the default file name in S3. If left empty the file name is a GUID with a timestamp. This field supports dynamic output references to control the key depending on the output payload.

Payload Reference

When working with complex payloads, this setting uses dynamic outputs to specify the attribute that contains the file payload (e.g. `{{this.filePayload}}`).

Note: *If left blank, the entire payload is written out.*

Metadata

Sets system defined name-value pairs included with the value sent to the S3 bucket. For example, users can set the **Content-Type** of the payload to different types such as **application/json** or **text/csv**.

Storage Class

Specify the storage class of the payloads being written out to S3.

Prefix Path with UTC Time

When enabled a time prefix is added to the key to logically separate files in S3 as `yyyy/MM/dd/HH/key`.

S3 Tables | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/amazon_s3_tables/

S3 Tables

AWS S3 tables provide tabular data storage on top of S3, allowing applications to treat S3 like a SQL data store. It does this by writing parquet files to S3 and updating the catalog to inform S3 tables what the schema of the files is.

Connection Settings

Access Key

IAM created user access key with S3 and S3 Tables permission to read and write.

Secret Key

IAM provided secret key.

Region

Region of the Amazon Kinesis Data Streams instance (e.g. `us-east-1`)

Input Settings

Inputs are not currently supported although the Athena JDBC driver can be used with the JDBC connection to read from S3 Tables

Output Settings

Table Bucket ARN

The full ARN of the table bucket to write to. For example: `arn:aws:s3tables:us-west-2:000000000000:bucket/test`

Namespace

The namespace to write to in the table bucket. Table buckets can have multiple namespaces but a table has a single namespace.

Table

The name of the table to write to. Table names must follow S3 naming restrictions (ex. no upper case).

Create

When Off is selected, the table and namespace must already exist. When Create is selected, the table and namespace are created if they don't exist. When Create & Update is selected the table and namespace are created if they don't exist, and the table schema are updated if new attributes appear in the write.

Output Examples

The S3 Tables connection writes out tabular data and matches the data to the table schema by name.

For example, assume the payload is as follows. Assume the table already exists, and has two columns "col1" and "col2".

```
{
  "col1": 1.23,
  "col2": "hello world",
  "col3": null
}
```

The above write will match col1 and col2 by name, and insert a single row into the table [1.23, “hello world”]. Following the example, assume the following payload is written to the same table, but Create & Update is enabled.

```
[
  {
    "col1": 1.23,
    "col2": "hello world"
  },
  {
    "col3": false
  }
]
```

This will add a “col3” to the table with a data type of boolean, and then insert the first row as [1.23, “hello world”, null] and the second row as [null, null, false].

Data Firehose | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/aws_data_firehose/

Data Firehose

The AWS Data Firehose connector uses the AWS SDK to communicate to Amazon Data Firehose, which is REST based. The Access Key and Secret Key must be acquired for an IAM user that is in a group with access to the Amazon Data Firehose API. This permission in AWS is called `AmazonKinesisFirehoseFullAccess`. This role is used to publish data to Amazon Data Firehose.

Connection Settings

Access Key

IAM created user access key with `AmazonKinesisFirehoseFullAccess` permission.

Secret Key

IAM provided secret key.

Region

Region of the Amazon Data Firehose instance (e.g., `us-east-1`)

Input Settings

Inputs are not currently supported

Output Settings

Delivery Stream

The name of the delivery stream to write to.

IoT SiteWise | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/aws_iot_sitewise/

IoT SiteWise

The AWS IoT SiteWise connector supports writing data to SiteWise. Models and Instances in the Intelligence Hub are converted to Models and Assets in SiteWise. The Connector supports browsing the SiteWise hierarchy and writing data in realtime using the SiteWise API as well as SiteWise Buffered Ingest using S3.

Connection Settings

Authentication Type

The type of authentication to use.

Setting	Description
Token	Enter an IAM Access and Secret Key that have permissions to write to SiteWise. See below section on IAM permission best practices.
STS	Enter the IAM Access and Secret Key to an STS (Security Token Service) IAM user that can assume an IAM role to write to SiteWise. See the AWS documentation on how to setup STS. STS authentication also requires a Role ARN and Session Name.

Region

Region of the AWS IoT SiteWise instance (e.g., `us-east-1`)

Buffered Ingest Settings

Buffered Ingest supports writing data to SiteWise through S3, which can be more cost-effective for large volumes of data that are not needed in real-time. The Buffered Ingest settings are used by Outputs that are set to `Buffered` mode. If Buffered Ingest is not being used, these settings can be ignored.

Setting	Description
S3 Bucket	The name of the S3 Bucket to write to. The account used for authentication must have access to write to this bucket.
S3 File Key	An optional prefix for the file names placed in the bucket. Files are named -<epoch_ms>.csv. If left blank the product name is used.
Delete S3 File	When enabled, SiteWise deletes the files from S3 after a successful upload.

Input Settings

Inputs are not currently supported

Output Settings

Parent Asset

The parent asset to write to. This must be the full path to the parent (ex. boston/seaport/building1). Use `Find` to browse for the asset. If left blank the write occurs at the root of the SiteWise hierarchy. If the parent is not found the write fails.

Hierarchy Name

The name of the parent Asset's hierarchy to write to. Use `Find` to browse for available hierarchies. As an example a "Region" model may have a hierarchy named "sites". When writing a new site the parent asset would be "North America" and hierarchy would be "sites".

Leave this blank when writing to the root. If the hierarchy name is not found and `Create` is enabled, the hierarchy is created on the parent Model. This field is only visible if the Parent Asset setting is not blank.

Create

When enabled Assets, Models, and Model hierarchy are created if they don't already exist. Create behaves as follows.

- Look up the Model by name. If it doesn't exist create a new Model
- If the Model is found, check to see if model definition has changed. If it has, update the model with the new attributes or hierarchy
- Look up the Asset using the full path (including parent hierarchy). If it doesn't exist create a new Asset

Create is additive only. Renaming, moving, or deleting of Assets, Models, or Attributes is not supported.

Asset Name

The name of the Asset to write. If left blank the Asset name is taken from the output data (ex. instance name, or "_name" attribute in the payload).

Model Name

The name of the Model to write. If left blank the Model name is taken from the output data (ex. model name or "_model" attribute in the payload).

Timestamp

The timestamp used for VQTs in the write. This can be taken from the payload using a dynamic reference. By default the timestamp comes from the source data. Timestamps can be in epoch millisecond or ISO 8601 format

Write Path

The write path controls how data is written to SiteWise. It does not impact Model and Asset creation.

When set to Realtime, the BatchPutAssetPropertyValue API is used. The Output attempts to batch writes into 10x10 chunks, where each write contains 10 measurements (i.e. VQTs), each with 10 values. This results in the fastest possible ingest time for realtime data, but can result in higher ingest cost. Buffering in a pipeline may be required to take advantage of the chunking feature.

When set to Buffered, writes are sent to S3 in CSV format. Each write is a unique CSV file. The output then calls the SiteWise CreateBulkImportJob API to load the file from S3. This can be more efficient for large writes, and can result in sub-minute load times. In general this ingest method is more efficient for large amounts of data that's not needed in realtime. SiteWise recommends 12,000 VQTs (i.e. data changes or rows in the CSV file) in a single file. This results in a file size of around 10MB.

Note careful consideration should be given to each write path type given data requirements. Using Buffered mode for small amounts of realtime data can be just as inefficient and costly as using Realtime mode for large amounts of non-realtime data.

A single Connection can support Outputs configured in both Realtime and Buffered mode. |

How Writes Work

The SiteWise Model and Asset information is taken from the payload. Below is an example write. This write could come from a raw input (ex. MQTT message), be generated manually, or be generated by the Models and Instances configured in the product. Regardless of the source, the SiteWise output translates the JSON below into SiteWise Assets and Models.

```
{
  "_name": "North_America",
  "_model": "Region",
  "regionCode": 123,
  "sites": [
    {
      "_name": "Portland",
      "_model": "Site",
      "address": "100 Congress Street"
    },
    {
      "_name": "Portsmouth",
      "_model": "Site",
      "address": "100 Main Street"
    }
  ]
}
```

There are two models in the above payload named Region and Site. Region has two Site Models linked by a “sites” label, and an attribute named “regionCode”. Site has a single attribute “address”. When this payload is written to an output the following happens.

1. The models Site and Region are looked up by name. If either doesn’t exist and create is off, the write fails. If they both exist and create is off, proceed to the next step. If create is on, the output creates the models if they don’t exist. If they do exist the output checks to see if the JSON schema is the same as the SiteWise model schema. If it’s different the output updates the models with new attributes or relationships. All attributes of the JSON that are not relationships (ex. sites is a relationship) are converted to Measurements in SiteWise. The output produces an error message “Building Models” and “Building Assets” until all Models and Assets in the JSON structure are built out.
2. The assets Portsmouth, Portland, and North_America are looked up by name. If create is off, and they don’t exist the write fails. If they do exist proceed to the next step. If create is on and they don’t exist they are created.
3. Once the Models and Assets exist in SiteWise, writes are issued for each attribute/measurement. Attributes are matched by name. For example, if the attribute “address” doesn’t exist on the Site model in SiteWise, it is skipped in the write. Writes for the entire JSON object are issue together.

Note that when create is enabled, all changes to Models and Assets are additive. The output will not delete or rename existing SiteWise Measurements or Hierarchy.

Asset and Model Cache

On the initial connection the SiteWise connector caches all Models and Assets in SiteWise. If Assets or Models are added by other systems after the connection, the user must re-save the connection reconnect and cache the state again.

AWS IAM Best Practices

Please see AWS documentation on IAM best practices. HighByte strongly recommends following the policy of least privilege when granting the IAM role for the connection.

https://docs.aws.amazon.com/iot-sitewise/latest/userguide/security-iam.html
https://docs.aws.amazon.com/iot/latest/developerguide/security-best-practices.html

It is also recommended that users occasionally rotate new IAM credentials and manually update the Intelligence Hub configuration with the new credentials.

The following IAM permissions are used by the SiteWise Connection.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
    {
          "Sid": "VisualEditor0",
          "Effect": "Allow",
          "Action": [
               "iotsitewise:CreateAsset",
               "iotsitewise:CreateAssetModel",
               "iotsitewise:ListAssets",
               "iotsitewise:ListAssetModels",
               "iotsitewise:ListAssociatedAssets",
               "iotsitewise:ListAssetRelationships",
               "iotsitewise:DescribeAssetModel",
               "iotsitewise:DescribeAsset",
               "iotsitewise:AssociateAssets",
               "iotsitewise:UpdateAssetModel",
               "iotsitewise:BatchPutAssetPropertyValue",
               "iotsitewise:CreateBulkImportJob",
               "iotsitewise:DescribeStorageConfiguration"
          ],
"Resource": "*"
    }
  ]
}
```

Buffered Ingest Setup

Buffered Ingest is not enabled by default in SiteWise. To enable Buffered Ingest you must do the following. See the SiteWise documentation for more details.

1. Login to the SiteWise console and go to Settings | Storage and enable “Activate Cold and Warm tier storage”
2. Setup an IAM role that has permission to write to SiteWise, the S3 Bucket, and the IAM:passRole permission
    - Below is an example of the passRole permission. This is required by the CreateBulkImportJob SiteWise API call

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "*"
      }
   ]
}
```

Kinesis Data Streams | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/aws/aws_kinesis_data_streams/

Kinesis Data Streams

The Amazon Kinesis Data Streams connector uses the AWS SDK to communicate to Amazon Kinesis Data Streams, which is REST based. The Access Key and Secret Key must be acquired for an IAM user that is in a group with access to the Amazon Kinesis Data Streams API. This permission in AWS is called AmazonKinesisFullAccess. This role is used to publish data to Amazon Kinesis Data Streams.

Connection Settings

Access Key

IAM created user access key with `AmazonKinesisFullAccess` permission.

Secret Key

IAM provided secret key.

Region

Region of the Amazon Kinesis Data Streams instance (e.g., `us-east-1`)

Input Settings

Inputs are not currently supported

Output Settings

Data Stream

The name of data stream to write to.

Partition Key

An optional field to specify a partition key for the data stream. Outputs with a specified partition key will all write to the same shard. If no partition key is given, the output will disperse requests among all existing shards for the data stream.

Microsoft Azure | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/azure/

Microsoft Azure

- Blob Storage
- OneLake
- Event Hubs
- IoT Edge
- IoT Hub
- Microsoft Fabric

Blob Storage | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/azure/azure_blob_storage/

Blob Storage

Connection Settings

Endpoint URL

The endpoint URL for the Blob Storage, found under the Blob Storage configuration in Azure.

Connection String

The primary connection string of the Blob Storage, found under the Blob Storage configuration in Azure.

Input Settings

Container Name

The name of the blob storage container to read data from.

Blob Name

The name of the blob to read. Note that the blob name may include the virtual directory path where applicable (e.g. `path/to/myBlobName.csv`).

Encoding

The character set used to decode the contents of the file. Select **Binary** to read the file as binary and skip decoding. Select **Auto** to decode the file using the BOM of the file. Otherwise, use a specific encoding type to decode the file.

Include Metadata

When set to true, includes properties specific to the Azure Blob. Includes both system defined properties along with any custom metadata key-value pairs defined on the blob.

Output Settings

Container Name

The name of the storage container to place the data. Supports dynamic references.

Blob Name

The name of the blob. If no name is provided the blob name is a GUID and timestamp. This field supports dynamic references.

Payload Reference

When working with complex payloads, this setting uses dynamic outputs to specify the attribute that contains the file payload (e.g. `{{this.filePayload}}`).

Note: *If left blank, the entire payload is written out.*

Prefix Path with UTC Time

When enabled a time prefix is added to the key to logically separate files in S3 as `yyyy/MM/dd/HH/key`.

Event Hubs | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/azure/azure_event_hubs/

Event Hubs

The Azure Event Hubs connection uses the AMQP protocol to connect to Event Hubs. This may require opening outbound ports 5671 (insecure) and 5672 (secure) in your firewall.

Connection Settings

Connection String

The primary connection string for the Event Hubs Namespace. Found under shared access policies in the Azure Console's Event Hubs view.

Protocol

Specifies the underlying protocol to use for connectivity. Options include `AMQP` and `AMQP Over Websockets`.

Event Hub Name

The name of the event hub to publish to.

Consumer Group

The name of the consumer group that subscribes to published events. If no consumer group is provided, the default consumer group is used.

Request Timeout

Maximum time to wait for a send request to respond before failing. Valid range is 100-60000 milliseconds.

Input Settings

Partition

The partition id to read from. If no partition id is provided, data from all existing partitions is read.

Include Metadata

Includes metadata specific to Azure Event Hub. When turned on the partition id of the payload is included.

Output Settings

Template

If blank, the default JSON output format is used. See Output Templates for details on the template syntax.

IoT Edge | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/azure/azure_iot_edge/

IoT Edge

Connection Settings

Protocol

Specifies the underlying protocol to use for connectivity. Options include AMQPS, HTTPS and MQTT.

Note: *Inputs will not function if the protocol is set to HTTPS.*

Input Settings

Sink

The name of the sink that the input reads from. This must match the routes configured in the Azure IoT Edge manifest file.

Output Settings

Source

The source to publish data to. This must match the routes configured in the Azure IoT Edge manifest file.

IoT Hub | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/azure/azure_iot_hub/

IoT Hub

Connection Settings

Protocol

Specifies the underlying protocol to use for connectivity. Options include AMQPS, HTTPS and MQTT.

Note: *Inputs will not function if the protocol is set to HTTPS.*

Provision Type

The type of provisioning used to authenticate and connect to the IoT Hub; Connection String or DPS (Device Provisioning Service). DPS is required when connecting to Azure IoT Central.

Connection String

The primary connection string of the IoT Hub's IoT Device, found under IoT devices in Azure Console's IoT Hub configuration.

DPS Scope ID

The scope ID for the DPS service used to provision connections to the IoT Hub. This is provided in the DPS configuration.

DPS Device ID

The Device ID that is unique for the DPS connection. This is provided by the DPS configuration.

DPS Key

The DPS Key used to authenticate to the DPS. This is provided by the DPS configuration.

DTDL Model ID

When publishing to Azure IoT Central, set this to the model ID for the DTDL model/template the intelligence hub is publishing to.

Input Settings

No input protocol specific settings are required

Output Settings

Template

If blank, the default JSON output format is used. See Output Templates for details on the template syntax.

Model Component

When publishing to Azure IoT Central using a DTDL data model and the model has subcomponents, set this to the subcomponent name to write to. This ensures writes are sent to the correct sub component.

OneLake | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/azure/azure_onelake/

OneLake

Connection Settings

Endpoint URL

The endpoint URL for the Data Lake storage, found under the Storage Account Endpoints in Azure.

Connection String

The primary connection string of the Gen 2 Storage account, found under Security + Networking Access Keys in Azure.

Output Settings

File System Name

The name of the Gen 2 storage container to place the data. Supports dynamic references.

File Path

The file path to use to place the data. If no name is provided the blob name is a GUID and timestamp. This field supports dynamic references.

Payload Reference

When working with complex payloads, this setting uses dynamic outputs to specify the attribute that contains the file payload (e.g. `{{this.filePayload}}`).

Note: *If left blank, the entire payload is written out.*

Prefix Path with UTC Time

When enabled a time prefix is added to the key to logically separate files in S3 as `yyyy/MM/dd/HH/key`.

Microsoft Fabric | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/azure/microsoft_fabric/

Microsoft Fabric

Microsoft Fabric is a new data platform from Microsoft that includes a number of the core Azure services and allows data engineering, integration, analytics, and intelligence all in one place, built on top of OneLake. Although Microsoft Fabric is new, a lot of the existing Azure services can be used to ingest data into Fabric. This page covers the different integration options.

Realtime Data

Real-Time Intelligence (RTI) in Microsoft Fabric focuses on realtime data ingestion, exploration, and analysis. It's the ideal entry point for realtime machine and telemetry data. The Intelligence Hub gets data into RTI using RTI's Eventstreams. Eventstreams is a generic term for the different integration options available in RTI for realtime data. The following options exist today.

Integration	Notes
Event Hubs	Event Hubs is the ideal integration and can be setup as an independent Event Hub that RTI consumes from or an Event Hub endpoint hosted in RTI. The Intelligence Hub publishes to the Event Hub and RTI consumes the events.
IoT Hub	This is similar to Event Hubs, but must be hosted as an independent IoT Hub service. IntelligenceHub publishes to IoT Hub and RTI consumes the events.
MQTT	This requires a 3rd party broker be configured. The IntelligenceHub publishes and RTI consumes.
Kafka	This requires a 3rd party Kafka cluster to be setup. The IntelligenceHub publishes and RTI consumes.
Other	See the latest RTI Eventstreams documentation for the latest RTI integrations.

Bulk Data

For bulk data ingestion, which can be historical or MES data, as well as buffered realtime data, Azure Blob storage is the ideal integration point. The Azure Blob connection can be used to upload the data to Blob or Gen2 storage, and RTI can access it from there.

Databricks Client | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/databricks_client/

Databricks Client

The Databricks Client connects to Databricks Workspaces using the Databricks REST API.

Connection Settings

Host

The host address of the Databricks instance. It should follow the format: `https://XXX-XXXXXXXX-XXXX.cloud.databricks.com`.

Authentication Type

Specifies the authentication method used to connect to the Databricks instance. Personal Access Token and OAuth are supported.

Personal Access Token (PAT)

Used to authenticate individual users.

Token

The token used for user authentication.

OAuth

Used to authenticate service principals and users.

Client ID

The client ID of the service principal or user.

Client Secret

The client secret of the service principal or user.

Request Timeout

The maximum number of seconds to wait for a response.

Input Settings

Inputs are not supported.

Output Settings

Catalog

The name of the catalog.

Schema

The name of the schema.

Volume

The name of the volume.

Path

The relative path of the file within the volume. Use forward slashes (`/`) to separate directories. The path should end with the file name.

Databricks SQL | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/databricks_sql/

Databricks SQL

The Databricks SQL Connection enables the ingestion of structured data from Databricks Workspaces using a SQL interface.

Connection Settings

JDBC Connection String

The JDBC connection string of the Databricks instance. It should follow the format: `jdbc:databricks://XXX-XXXXXXXX-XXXX.cloud.databricks.com:443`.

HTTP Path

The HTTP path of the SQL warehouse. It should follow the format: `/sql/1.0/warehouses/XXXXXXXXXXXXX`.

Authentication Type

Specifies the authentication method used to connect to the Databricks instance. Personal Access Token and OAuth are supported.

Personal Access Token (PAT)

Used to authenticate individual users.

Token

The token used for user authentication.

OAuth

Used to authenticate service principals and users.

Client ID

The client ID of the service principal or user.

Client Secret

The client secret of the service principal or user.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

See JDBC Driver Settings for input settings.

Output Settings

Outputs are not supported.

Files | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/files/

Files

- CSV
- File
- Parquet

CSV | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/files/csv/

CSV

The CSV Connector supports reading and writing CSV files.

Connection Settings

File Directory

Specifies the root directory for file processing.

Processed File Directory

Specifies the directory where files are moved to after processing successfully. This is only used for inputs that enable “Process File”.

Error Directory

Specifies the directory where files are moved to after processing unsuccessfully. This is only used for inputs that enable “Process File”.

Input Settings

CSV File Name

The file name to read. This can be the full name of a single file or a regular expression. As an example, (.*).csv would match the first .csv file in the directory based on alphabetical order.

Relative Path

The relative path of the starting directory in regard to the connection level File Directory setting.

File Encoding

The encoding of the CSV file required to properly decode the file. Select Auto if the file has the correct BOM (byte order mark). In cases where the file has no BOM or an incorrect BOM, override the decoding by selecting one of the available options.

Skip Lines

How many lines to skip from the front of the file. Use this setting to skip header lines or other data that is not required.

Max Rows

The max number of rows in the file. Leave this blank if you just want to read to the end of the file. Set this if you only want to read up to N rows.

Max Rows Per Read

The max number of rows to return from a single read. As an example, if there are 1000 rows and this is set to 10, only 10 rows would be read at a time. When Index File is on, the first read will be records 0 to 9, second read will be 10 to 19, etc. When left blank, this defaults to 100.

Replacement Headers

Delimited separated list of headers used to replace the file headers, or provide headers (e.g., header1, header2, header3).

Note: There must be a header for each column of data.

Delimiter

Specifies the delimiter used to separate headers and fields. The default delimiter is a comma. Only single character delimiters are supported.

Quote

The quote character used by the CSV parser. Defaults to ". Supports the following escaped characters \t (tab), \r (return), \n (newline). Leave this blank to disable quoting.

Escape

The escape character used by the CSV parser. Defaults to \. Supports the following escaped characters \t (tab), \r (return), \n (newline). Leave this blank to disable escaping.

Process File

When enabled, the file is moved to the connection's processed directory after being read.

Output Settings

File Name

Specifies the name of the file to write to. This includes the file extension and directories. The file path can include dynamic outputs (e.g., `/{{this.site}}/file.csv`)

Relative Path

The relative path of the starting directory in regard to the connection level File Directory setting.

Delimiter

The delimiter to use in the output file. By default, this is a comma.

Create

When enabled, the file and directory structure are created if they do not exist. If they do exist, data is appended to the end of the file.

File | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/files/file/

File

The File Connector supports reading and writing files locally on the file system, via SMB (Server Message Block), FTP, and SFTP.

Connection Settings

Communication Protocol

Specifies the communication protocol used to access the files. The following options are supported.

Option	Description
File Protocol	Used for local directory access. Local in this context means on the same machine the Intelligence Hub is running on.
SMB	Used to access shared network directories.
FTP	Access files on an FTP server.
SFTP	Access files over SFTP.

Communication Protocol: SMB

The following connection settings are available for the SMB protocol.

Setting	Description
Share	Specifies the server and share name of the access shared network directory. Ex. \ServerName\ShareName
Domain	Specifies the domain of the server used to access the shared network directory.
Username	Username required to authenticate with the share. Leave this blank for anonymous access.
Password	Password required to authenticate with the share. Leave this blank for anonymous access.

Communication Protocol: FTP

The following connection settings are available for the FTP protocol.

Setting	Description
Server Type	In most cases this should be set to Unix. In some cases, Windows specific FTP servers may need the Windows option selected.
Use Passive Mode	In most cases passive mode should be used. Active mode requires the FTP server to connect back to the FTP Client, and is more difficult to configure securely.
Host	Specifies the hostname of the FTP server.
Port	Specifies the port of the FTP server.

Setting	Description
Username	Username required to authenticate with the share. Leave this blank for anonymous access.
Password	Password required to authenticate with the share. Leave this blank for anonymous access.

Communication Protocol: SFTP

The following connection settings are available for the SFTP protocol.

Setting	Description
Host	Specifies the hostname of the SFTP server.
Port	Specifies the port of the SFTP server.
Connect Timeout	The duration, in seconds, that the file connection will allow the SFTP host to respond during a connection attempt before aborting the attempt.
Session Timeout	The duration, in seconds, that the file connection will allow the SFTP host to respond before aborting the request.
Username	Username required to authenticate with the server.
Password	Password required to authenticate with the server.

Common Connection Settings

The following settings are applicable to all protocols.

Setting	Description
File Directory	Specifies the root directory for file processing.
Processed File Directory	Specifies the directory where files are moved to after processing successfully. This is only used by inputs that enable “Process File”.
Error Directory	Specifies the directory where files are moved to after processing unsuccessfully. This is only used by inputs that enable “Process File”.

Browse

Input and Output Browsing

After the connection has been configured, the browse button can be used to navigate through the file system and create an Input or Output for any file or directory.

Input Settings

File Input Type

The File input type returns the content of a single file and has the following settings

File Search Type

How the file should be found. This can either be a direct search based on the full path of the file, or a regex search for a file name with an incomplete path.

Full Path Search Type

The full path search type allows a user to read a file when the full path to the file is already known. Due to the reduced searching involved, the full path search type will generally be more performant especially in large file systems. When a file is not found with this search type, an error is thrown.

File Path

The full path the a file in the file system, including its name.

Regex Search Type

The regex search type allows a user to find a file which matches a regex expression. It primary use is for reading a file or set of files when names, locations, or existence is unknown. When no file is found, nothing is returned and no error is thrown.

File Regex

The file name to read. This can be the full name of a single file or a regular expression. As an example, (.+) matches any file in the directory.

Relative Path

The relative path of the starting directory in regards to the connection level File Directory setting.

File Encoding

The character set used to decode the contents of the file.

Process File

When enabled, the file is moved to the connection's processed directory after being read.

Include Metadata

Include metadata in the read, including file name, relative file path, creation and update times (if available), and file size.

Directory Input Type

The Directory input returns a list of the contents of a directory and has the following settings

Relative Path

The relative path of the starting directory in regards to the connection level File Directory setting.

Recursive

Designates whether the directory input should recursively explore sub directories.

Common settings

Time Filter

The time filter can be set to filter on either creation or update time, and will cause the input to return only files or directories which have been either created or updated after a provided ISO 8601 time.

Note: *Time filter is dependent on both the protocol and server operating system. Protocols such as FTP and SFTP which don't support creation time will not be able to be filtered on creation.*

Output Settings

File Name

Specifies the name of the file to write to. This includes the file extension and directories. The file path can include dynamic outputs (e.g., `/{{this.site}}/file.bin`)

Relative Path

The relative path of the starting directory in regard to the connection level File Directory setting.

Payload Reference

When working with complex payloads, this setting uses dynamic outputs to specify the attribute that contains the file payload (e.g. `{{this.filePayload}}`).

Note: *If left blank, the file connector will write the entire payload out to a file.*

Base64 Decoding

When enabled, payloads are base64 decoded before being output as a file. This is used in when receiving Base64 encoded file payloads over other connections such as REST or MQTT.

Parquet | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/files/parquet/

Parquet

Connection Settings

File Directory

Specifies the directory where files should be sourced for processing.

Processed File Directory

Specifies the directory where files should be moved to after processing successfully. This is only used for inputs that enable "Process File".

Error Directory

Specifies the directory where files should be moved to after processing unsuccessfully. This is only used for inputs that enable "Process File".

Input Settings

File Name

The file name to read. This can be the full name of a single file or a regular expression. As an example, `(.*).parquet` would match the first `.parquet` file in the directory based on alphabetical order.

Relative Path

The relative path of the starting directory in regard to the connection level File Directory setting.

Max Rows Per Read

The max number of rows in the file. Leave this blank if you just want to read to the end of the file. Set this if you only want to limit the rows to read.

Process File

When enabled, the file is moved to the connection's processed directory after being read.

Output Settings

Base File Name

The base filename to write out. This field supports dynamic outputs to control the file name based on the output data. A timestamp and guid is also included with the file name to ensure uniqueness. The format of this string is `yyyy-MM-dd-HH-mm-ss-guid`.

Relative Path

The relative path of the starting directory with respect to the connection level File Directory setting.

File Duration (minutes)

Specifies how much data is in each file. The default duration is 1 minute. If set to 1 minute, each file contains one minute of data.

Google | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/google/

Google

- BigQuery
- PubSub

BigQuery | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/google/google_bigquery/

BigQuery

Connection Settings

The Google BigQuery connection uses the Simba BigQuery JDBC driver to enable reading, writing, and browsing BigQuery datasets. The connection uses the host `https://www.googleapis.com/bigquery/v2` on port `443` for BigQuery.

To use this connection, you must do the following.

1. Download the JDBC Driver linked to on this page: https://cloud.google.com/bigquery/docs/reference/odbc-jdbc-drivers
   - This release has been tested with version 1.5.2.1005 of the Simba driver
2. Unzip and copy the `GoogleBigQueryJDBC42.jar` into the runtime/lib directory.
   - The Intelligence Hub installer contains all the required dependencies included in the zip with the exception of the Simba driver which cannot not be redistributed with applications.
3. Restart the runtime

Below are the required setting for the Connection.

Project Id

The Project Id as defined in the Google Cloud Console. Login to BigQuery and create a project to get the Id. An example Id is `myproject-417313`

JDBC Connection String

Specifies additional arguments for the end of the JDBC connection string. For example, it's common to authenticate with BigQuery using a Service Account. A key is generated from the Service Account in the form of a JSON file. Place the JSON file on disk, making it available to the runtime, and then reference it in the JDBC Connection String as follows.

`OAuthType=0;OAuthServiceAcctEmail=serviceName@emailDomain;OAuthPvtKeyPath=C:/path/to/project-keyfile.json;`

To find additional support parameters, reference the Simba **Installation and Configuration Guide** for the Google BigQuery JDBC driver.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input/Output Settings

See JDBC Driver Settings for input and output settings.

Note: When specifying output tables, prefix the table with the name of the dataset. For example `dataset.table`.

PubSub | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/google/google_pubsub/

PubSub

Connection Settings

Project Id

The Project Id as defined in the Google Cloud Console.

Service Account JSON

The Google Console Service Account Key JSON file, used to grant access to the Service Account that has permission to publish to the Project and Topic.

Paste the contents of the key .json file directly in this field. As an example, the file should start with:

```
{
    "type": "service_account",
    "project_id": "myproject-312317",
    "private_key_id": "12345...",
}
```

Input Settings

Inputs are not currently supported

Output Settings

Topic

The Topic ID as defined in the Google Cloud Pub/Sub configuration. This is the topic the output will publish data to.

Historians | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/historians/

Historians

- AspenTech IP.21
  - AspenTech IP.21 Agent
- InfluxDB
- PI System
  - PI System AF SDK Connector
- TimescaleDB

AspenTech IP.21 | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/historians/aspentech_ip_21/

AspenTech IP.21

Connection Settings

IP

IP address or host name for the IP.21 Agent.

Port

Port for the IP.21 Agent.

Connection String

The ODBC Connection String used by the agent to connect to the InfoPlus server. The default string assumes the agent is running on the same machine as the InfoPlus server and establishes an local ODBC connection.

Default: `Driver={AspenTech SQLplus};Host=localhost;Port=10014`

When the agent is running on a separate machine this string should be updated use the `Host` and `Port` required to establish a remote ODBC connection with the server.

Token

The connection token used by the agent to authenticate the Intelligence Hub connection. This should match the token configured in the ip21 agent settings.json.

Connect Timeout (seconds)

The maximum amount of time to wait for successful connection to the agent. If this time is exceeded the connection fails with an error return.

Request Timeout (ms)

The maximum amount of time to wait for a read or write response. If this time is exceeded the read or write fails with an error return.

Compression

When enabled, communication between the Intelligence Hub and the agent is compressed using GZip. This improves read times in cases where returned payloads are large. For smaller payloads, there may be little or no performance improvement.

Input Settings

Type

`Query` is currently the only supported input type.

Command

The SqlPlus command to execute when the input is read. The Record Explorer can be used to discover record and attribute names available for use in SqlPlus queries.

NOTE: The Record Explorer does not expose virtual tables such as `HISTORY` or `ALL_RECORDS`. But these records can be used in SqlPlus commands.

AspenTech IP.21 Agent | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/historians/aspentech_ip_21/agent/

AspenTech IP.21 Agent

The AspenTech InfoPlus 21 Agent uses SqlPlus ODBC Driver to execute SqlPlus queries to read data from an InfoPlus server. The agent is a .NET application and is delivered separately from the core HighByte Intelligence Hub installation. The requirements for the agent are as follows.

1. Windows Operating System
2. Microsoft .NET 4.8 or greater
3. SqlPlus ODBC Driver. This is installed as part of the InfoPlus 21 installation or manually following instructions documented in the Aspen Support Knowledge Base.

The agent runs as a standalone application, by running the *intelligencehub-ip21.exe* OR as a Windows service. When running standalone, it's recommended to run the executable with admin level permissions. To install as a service, run the following command. InstallUtil is found in the .NET installation directory: `%windir%\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe`

```
installutil intelligencehub-ip21.exe
```

NOTE: Whether the agent is run in a console or a service the application should be run with a user that has read-only access to InfoPlus.21 database.

To optionally control the service name, using the following syntax.

```
installutil /serviceName="myServiceName" /displayName="myDisplayName" intelligencehub-ip21.e
```

The agent opens a port to receive requests from the HighByte Intelligence Hub instance. By default, this communication occurs using secure websockets. On the first run the agent creates a self-signed certificate called `intelligencehub.pfx` as well as a settings.json file. The file contains the following.

```
{
  "port": 45390,
  "token": "randomtoken"
}
```

The port controls the port that the agent listens on. This port must be open and available to the Intelligence Hub connecting to the agent. The token is a random security token created by the agent. This token must be applied to the Intelligence Hub IP.21 connection settings to authenticate the hub. Any changes to these settings require restarting the agent.

In the Intelligence Hub IP.21 connection settings, enter the IP/port of the agent, as well as the token. The agent also has a local `intelligencehub.log` file for debug information. See below on how to enable detailed diagnostics logging.

Diagnostics Logging

The IP.21 Agent includes detailed diagnostics to analyze read and write performance. The agent uses NLog for to control log retention policy and verbosity. To control diagnostics change the `intelligencehub-ip21.exe.config` file in the same directory as the `intelligencehub-ip21.exe` file.

Add the following configuration to the file, save the file, and restart the agent. Diagnostics logs will be included in the console output and in the log file.

If the agent is running as a Windows Service under a Windows account with access to IP.21, it must also have local file system access to write the diagnostics file locally. Otherwise, the local log file will not be written.

The default logging configuration is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- Register NLog as a configuration section -->
    <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog" requirePermission="fa
  </configSections>
  <nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <!-- Define targets (where logs are written) -->
    <targets>
      <!-- File target 3 files up to 100MB each -->
      <target xsi:type="File"
          name="logfile"
          fileName="intelligencehub.log"
          archiveFileName="intelligencehub.{#}.log"
          archiveNumbering="Rolling"
          archiveAboveSize="104857600"
        maxArchiveFiles="3"
        layout="${date:format=dd-MM-yyyy hh\:mm\:ss} | ${level} | Agent | ${message}" />
      <target xsi:type="Console" name="console" layout="${date:format=dd-MM-yyyy hh\:mm\:ss}
    </targets>

    <!-- Define rules (what gets logged and where) -->
    <rules>
      <!-- Log all levels to the file -->
      <logger name="*" minlevel="Trace" writeTo="logfile" />
      <logger name="*" minlevel="Trace" writeTo="console" />
    </rules>
  </nlog>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8"/>
  </startup>
</configuration>
```

InfluxDB | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/historians/influxdb/

InfluxDB

Connection Settings

URL

Specifies the URL to connect to. This is the base URL to the InfluxDB API.

Token

Specifies the token provided by InfluxDB to authenticate the connection. This token limits what HighByte Intelligence Hub has access to in InfluxDB.

Organization Name

The organization name in Influx. This is optional, and when left blank the default organization for the account is used.

Input Settings

Flux Query

The Flux query string used to query InfluxDB. See the InfluxDB documentation on supported query strings and syntax.

https://docs.influxdata.com/influxdb/v1.8/guides/query_data/

Data is returned in CSV format, and this data is available in HighByte Intelligence Hub as an array of rows.

Output Settings

Bucket

Specifies the the InfluxDB bucket to write data to. This bucket must exist in InfluxDB and be accessible by the connection token.

Measurement

Specifies the name measurement name. If left blank, the model or data type name is used.

This field supports dynamic outputs.

Name

Specifies the name tag added to the measurement. If left blank, the input or instance name is used.

This field supports dynamic outputs.

Tags

The tags associated with the data written to InfluxDB. Tags are a way to organize data in InfluxDB. Each tag is placed on a newline. As an example, tags can be used with dynamic outputs to organize data in an ISA-95 structure.

Timestamp

Specifies the timestamp of the data written to InfluxDB. If left blank, the timestamp of the payload is used. To set the timestamp either provide a UTC ISO-8601 format timestamp or the UTC epoch value in milliseconds.

This field support dynamic outputs.

PI System | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/historians/pi_system/

PI System

Connection Settings

Token

The connection token used by the PI System AF SDK Agent to authenticate the Intelligence Hub connection.

Connect Timeout (seconds)

The maximum amount of time to wait for successful connection to the agent. If this time is exceeded the connection fails with an error return.

Request Timeout (ms)

The maximum amount of time to wait for a read or write response. If this time is exceeded the read or write fails with an error return.

Compression

When enabled, communication between the Intelligence Hub and the PI Agent is compressed using GZip. This improves read times in cases where returned payloads are large. For smaller payloads, there may be little or no performance improvement.

PI Agent Authentication

In cases where the PI Agent and the PI Server are on the same Windows Domain, leave this as Windows. In Windows mode the PI Agent authenticates with the PI Server using the Windows credentials used to run the Agent. In cases where the Agent and Server are on different domains use Credentials mode. This mode passes the provided username/password to the PI Server for authentication.

Input Settings

Type

Specifies the type of data to read. The different types are described below.

Read Type	Description
Asset	Reads the values for one or more assets in AF given an AF query. This includes current value, historical, and more.
Asset Metadata	Reads the metadata for one or more assets in AF given an AF query. Metadata includes template definitions, attribute data types, point mappings, and more.
Event Frame	Reads one or more event frames from AF given an AF query.
Point	Reads point data from the archive. This includes current, historical data, raw data, averages, and more.
Point Change	Creates a DataPipe for one or more points, and returns new values, changed values, deleted, and more.
Point Browse	Returns point names and metadata given a PI point query.

Type: Asset

The following settings are available for Asset reads. See the Time Settings below for details on historical queries.

Setting	Description
Database	The AF Database to read from.
Query	The AF query to run against the AF Database. This supports the full PI AF syntax used in PI System Explorer. For example `PlantB\\Boilers` returns all Assets under this part of the tree. `Template:Boiler` returns all Assets backed by the Boiler template.
Query Paging	Enable paging of the query results. By default all assets in the query are returned. Paging allows you to return part of the result, improving performance. For example, a Page Index of 0 and Page Count of 10 returns the first 10 Assets in the query. An Index of 10 and Count of 10 returns the next 10. If there are no assets in the page, the input returns good no data.
Get	The type of query to make. Options include Current Value, Interpolated, and Raw Values. Current Value returns the current value of the assets, including all all attributes. Interpolated returns the interpolated values over time, and raw values returns the values at different periods of time. See below for details on the start/end time settings.
Attributes	For asset reads, select a subset of attributes to return in the results. When left blank all attributes are returned. This is helpful to speed up queries when assets have many attributes and only a few are required.
Include Metadata	Enable this to include information for each asset like the template name, timestamp, etc.
Include Children	Enable this to recursively include all child assets in the result from the Query and Query Page settings.

Type: Asset Changes

Subscribe to asset changes from the AF Server. This includes new data, changed data, deleted, and late arriving data. The first read browses the assets and all their attributes, and creates a subscription on the PI Agent for each attribute. Sub-sequent reads return point changes since the last read was performed.

Setting	Description
Database	Same as the Asset read.
Query	Same as the Asset read.
Subscription ID	A unique ID used to track the subscription on the PI Agent. This way the PI agent knows how to update the subscription if the point list changes. The ID should be unique per input and should not be shared across installations.

Type: Asset Metadata

The following settings are available for Asset Metadata reads.

Setting	Description
Database	The AF Database to read from.
Query	The AF query to run against the AF Database. This is the same as the Asset Query.

Query Paging, Include Children, and Include Attributes work the same as the Asset input. See the Asset input for details.

Type: Event Frame

The following settings are available for Event Frame reads.

Setting	Description
Database	The AF Database to read from.
Query	The AF query for the Events. This supports the full AF query query syntax available in System Explorer. For example `\*` returns all events, and `name:Boiler\*` returns all events named Boiler.

Query Paging, Include Children work the same as the Asset input. See the Asset input for details.

Type: Point

The following settings are available for Point reads. See the Time Settings below for details on historical queries.

Setting	Description
Points	A list of PI points to read.
Use Reference	When enabled, PI points are read from another data source like a CSV file or an Asset Meta Query. For example, a PI Point read can be setup to read all PI Points referenced under a given branch of the AF Hierarchy. The reference must return an array of strings, where each string is a point name (ex. `['point1', 'point2']`). An example of a Custom Condition is included below to read points from an Asset Metadata Query.
Get	The type of query to make. Options include Current Value, Raw Values, and Aggregates.
Include Metadata	Includes the following metadata for a point in the read result: pointID, descriptor, pointtype, engunits, span, zero, ptclassname, pointsource.
Additional Properties	Allows additional user defined metadata to be included in the read result when the read type is set to *Current Value*.

Type: Point Changes

Subscribe to point changes from an archive. This includes new data, changed data, deleted, and late arriving data. The first read creates a subscription on the PI Agent for the points, and sub-sequent reads return point changes since the last read was performed.

Setting	Description
Points	Same as the Point read.
Use Reference	Same as the Point read.
Subscription ID	A unique ID used to track the subscription on the PI Agent. This way the PI agent knows how to update the subscription if the point list changes. The ID should be unique per input and should not be shared across installations.
Inactive Timeout MS	This controls how long the subscription is left open with no reads before it is closed by the PI Agent. This exists to prevent the PI Agent from queueing changes forever and running out of memory. This should generally be set to twice the read interval for the input.

Type: Point Browse

The following settings are available for Point Browse read types.

Setting	Description
Query	The PI syntax query used to retrieve one of more points. For example SIN* returns everything that starts with “SIN”. This is the same syntax supported in native PI tools.
Include Metadata	When disabled, the browse result is an array of point names. When enabled, information like the description, data type, unique ID, etc are returned per point.

Setting	Description
Additional Properties	Allows additional user defined metadata to be included in the browse result.

Time Settings

These settings apply to Asset and Point reads, and allow for running queries over a large time range and reading the data in chunks.

Setting	Description
Start Time	Specifies the start time for the query. This uses the standard PI time syntax. Relative and absolute times are supported (ex. `*-1h`). Absolute times must include a `Z` at the end to be treated as UTC.
End Time	Specifies the end time for the query. The supported formats are the same as Start Time.
Interval	Specifies the interval to interpret data samples. For example, 5m, means every 5 minutes. Use this setting to break reads into chunks. As an example, get the average value of a point every 5 minutes over a 1-hour period When left blank the interval defaults to the time range between Start and End Time.
Time Index	When enabled, reads are performed from the Start to the End Time, in window intervals. After each successful read, the last read index is updated and stored to disk so that in the case of a system restart the reads start from where they last ended. Indexing is useful when querying a large amount of historical data.
Index Window	Used when indexing is enabled. This is the window size to read on each read request. As an example, with the following settings each read gathers one days' worth of data in 5 minute intervals and reads stop after the full 100 days is read.

Below is an example configuration. This reads data from 100 days ago until now, in 5 minute intervals, where each read returns one day of data.

```
Start Time = \*-100d
End Time = \*
Interval = 5m
Time Index = true
Window Size = +1d
```

Read Points from an Asset Meta Query

Below is an example of Custom Condition code that converts an Asset Meta query into a point list that can be used in a Point read. Using this technique you can read all the points referenced in an AF query, and detect new points when they are added. Note that asset queries are slower than point queries, so it's recommended to enabled input caching on the Asset Meta query. This avoids making Asset queries for every raw read.

Create a custom condition that references the Asset Meta query, and run it through this condition.

```
var assets = {{this.currentValue}}
var pointsOut = new Set();

// Get the point name for all attributes assigned to a point.
for (asset of assets){
    if ('attributes' in asset){
        for (attribute of asset.attributes){
            if ('point' in attribute){
                pointsOut.add(attribute.point.name)
            }
        }
    }
```

```
}

Array.from(pointsOut);
```

Output Settings

Type

Specifies the type of write; options include Asset and Points.

Point

Point writes write data as PI points. Arrays and simple types are not supported. Hierarchy is flattened using the `parent.child.attribute` syntax.

Setting	Description
Path	Specifies a prefix for the PI points. For example, "myPrefix" will write points as "myPrefix.pointName"
Point Name	The point name. If left blank, the point name is derived from the keys in the data.
Timestamp	Provides a way to set the timestamp of the write. When left blank, the timestamp will be taken from _timestamp in the outgoing payload. If no timestamp is provided and the payload doesn't contain _timestamp, the timestamp is set to the time of the write. This setting is used to pull the timestamp from the payload. As an example, `{{this.mytimestamp}}` pulls the timestamp from a `mytimestamp` attribute in the data. The timestamp must be in ISO8601 format.
Point Delimiter	Sets the delimiter used when flattening point names. By default it's "."
Update Option	Controls the API call made to insert data into the archive. All PI options are available.
Create	When enabled, points are created if they are not found.
Point Source	The source assigned to created points. By default this is HB.

Asset

Asset writes data as PI points, and then builds the templates and element hierarchy inside of PI's Asset Framework.

Setting	Description
Database	The Database to write the AF Assets to.
Path	Specifies the AF path to write to. For example `Portland\Shipping` writes to the shipping branch.
Asset Name	Overrides the element name assigned to elements. By default, when left blank, the element name is taken from the data (ex. Instance name).
Timestamp	Same as Point writes.
Point Delimiter	Same as Point writes.
Update Options	Same as Point writes.
Create	When enabled Assets are created in AF.
Point Source	Same as Point writes.

Note that if Create is enabled the output updates existing Elements and Templates if there are schema change. All updates are additive, meaning that renaming an attribute will keep the old attribute and add the new one.

PI System AF SDK Connector | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/historians/pi_system/pi-system-af-sdk-connector/

PI System AF SDK Connector

The PI System AF SDK Connector uses the PI System AF SDK to communicate with the PI and AF Server. The agent is a .NET application and is delivered separately from the core HighByte Intelligence Hub installation. The requirements for the agent are as follows.

1. Windows Operating System
2. The PI System AF SDK/AF Client installed on the system running the agent
3. Microsoft .NET 4.8 or greater

The agent runs as a standalone application, by running the *intelligencehub-osisoft.exe* OR as a Windows service. When running standalone, it's recommended to run the executable with admin level permissions. To install as a service, run the following command. InstallUtil is found in the .NET installation directory:
`%windir%\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe`

```
installutil intelligencehub-osisoft.exe
```

To optionally control the service name, using the following syntax.

```
installutil /serviceName="myServiceName" /displayName="myDisplayName" intelligencehub-osisof
```

The agent connects to the AF SDK/AF Client and opens a port to receive requests from the HighByte Intelligence Hub instance. By default, this communication occurs using secure websockets. On the first run the agent creates a self-signed certificate called `intelligencehub.pfx` as well as a settings.json file. The file contains the following.

```
{
  "port":45290,
  "token": "randomtoken"
}
```

The port controls the port that the agent listens on. This port must be open and available to the Intelligence Hub connecting to the agent. The token is a random security token created by the agent. This token must be applied to the Intelligence Hub PI System connection settings to authenticate the hub. Any changes to these settings require restarting the agent.

In the Intelligence Hub PI System connection settings, enter the IP/port of the agent, as well as the token. The agent also has a local `intelligencehub.log` file for debug information. See below on how to enable detailed diagnostics logging.

Note the agent connects to the default PI Archive and Asset Framework system configured by the AF Client using the PI System Explorer.

Diagnostics Logging

The PI Agent includes detailed diagnostics to analyze read and write performance. The agent uses NLog for to control log retention policy and verbosity. To control diagnostics change the `intelligencehub-osisoft.exe.config` file in the same directory as the `intelligencehub-osisoft.exe` file.

Add the following configuration to the file, save the file, and restart the agent. Diagnostics logs will be included in the console output and in the log file.

If the agent is running as a Windows Service under a Windows account with access to PI, it must also have local file system access to write the diagnostics file locally. Otherwise, the local log file will not be written.

The default logging configuration is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- Register NLog as a configuration section -->
    <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog" requirePermission="fa
  </configSections>
  <nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <!-- Define targets (where logs are written) -->
    <targets>
      <!-- File target 3 files up to 100MB each -->
      <target xsi:type="File"
          name="logfile"
          fileName="intelligencehub.log"
          archiveFileName="intelligencehub.{#}.log"
          archiveNumbering="Rolling"
          archiveAboveSize="104857600"
        maxArchiveFiles="3"
        layout="${date:format=dd-MM-yyyy hh\:mm\:ss} | ${level} | Agent | ${message}" />
      <target xsi:type="Console" name="console" layout="${date:format=dd-MM-yyyy hh\:mm\:ss}
    </targets>

    <!-- Define rules (what gets logged and where) -->
    <rules>
      <!-- Log all levels to the file -->
      <logger name="*" minlevel="Trace" writeTo="logfile" />
      <logger name="*" minlevel="Trace" writeTo="console" />
    </rules>
  </nlog>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8"/>
  </startup>
</configuration>
```

TimescaleDB | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/historians/timescaledb/

TimescaleDB

Connection Settings

Database

Name of the database to connect to.

Username

Username for authentication with the database.

Password

Password for authentication with the database.

Additional JDBC Options

Additional JDBC options to include in the JDBC connection string. This is PostgreSQL specific. Options are entered exactly as they would be if the JDBC connection string was created manually.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

Query

The full SQL query, which can include any SQL syntax including stored procedures.

Index

This setting is used to index the SQL query and only get new data on each query. When enabled, specifies a Column Name and default Value to reference within the query. This name/value pair is cached across successive product runs and allows the next query to pick up where it left off.

Note the Column Name must be the column name in the table that is being used for indexing. An example query might be SELECT * FROM SimpleMachineX WHERE id > {{index}} By enabling indexing and specifying the Column Name as id and the Value as zero, the first time the query is run {{index}} is set to zero and after the query {{index}} is updated to the latest value from id. The latest value is cached to disk. Successive queries will use the latest value cached and then update the cached value on success.

To reset the index, navigate to the “Index” page and delete the index value. Note that test reads on the input done from the UI will use the current index Value but will not update the index Value.

Column Name

Specifies the column name used to index. To update the index, the column name must be returned in the SQL query.

Value

Specifies the initial value used for the index.

Output Settings

Table

The name of the SQL table or stored procedure to write to. The table must already exist and have column names that match the model attribute names. Table names containing capital letters must be placed inside quotes (e.g., "MyTable").

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are **Off**, **Create**, and **Create & Update**.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. Non-Hypertables are created with an auto incrementing _id column used as the primary key. All Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 2048 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR(2048)
DateTime	DATETIME
UInt8	SMALLINT
Int8	SMALLINT
UInt16	INTEGER
Int16	SMALLINT
UInt32	BIGINT
Int32	INTEGER
UInt64	NUMERIC

Write Data Type	SQL Data Type
Int64	BIGINT
Real32	REAL
Real64	FLOAT

Create Hypertable

Specifies whether created tables should be created as Timescale Hypertables. Hypertables will not be created with the _id key, and do not have primary keys. Other metadata keys such as _name, _model, and _timestamp are still present.

Note: A minimum of Timescale version 2.13 is required for Hypertable creation.

Partition By

Specifies the column on which the Hypertable should be partitioned. This column must be of type timestamp, timestampz, date, or int, but timestampz is preferred per Timescale documentation. Hypertables with sub-partitioning must be made directly through Timescale.

Partition Interval

Specifies the size of chunks that Timescale should partition the database into.

Primary Key

Specifies a primary key for the created Hypertable. If no primary key is specified, the table will have no primary key. If a primary key is specified, both the primary key and the partition column will be used a primary keys for the created table. The same column may be specified as both the partition column and primary key.

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when Create Table is set to Off. Disable this refresh by setting the time interval to 0.

Ignition Module | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/ignition_module/

Ignition Module

The Ignition Module connection allows for browsing, reading, and writing tag data and UDTs to and from Ignition. The connection requires that the HighByte Ignition Module be installed on the Ignition server. For details on downloading and installing the module, please see the install section.

With the module installed in Ignition, the Intelligence Hub connects securely to the module using secure web sockets.

Connection Settings

Host

The IP address or domain name of the Ignition server.

Port

The port used to connect. By default the port is 45280. This port must be open and available on the Ignition server.

Password

The password used to authenticate the connection. This is set in Ignition under the module configuration (see module installation).

Connect Timeout (seconds)

How long to wait for a connection before failing.

Request Timeout (ms)

How long to wait for a read or write to complete before failing.

Input Settings

Inputs can be tags, folders, or UDTs from any Tag Provider in Ignition. Input reads are subscribed and serviced from cache during read. If a Tag Provider is changed, for example a tag is added or deleted, the module automatically detects this change and reflects the changes in the next read.

All input types can be used as Event sources. When used as an event source data updates are changed-based using Ignition subscriptions. The `_name` and `_model` keys are always included with UDT instances for identification purposes. Reading an input used as an event source returns the **latest cached** data associated with that address. The Ignition Module buffers tag changes in 100ms intervals to produce data change events. Data changes are not lost unless the journal limit of 10K is exceeded in any given 100ms interval. Event log message are logged any time the data change journal overflows.

Tag Address

The full tag address, as defined by Ignition. This includes the Tag Provider and the path. The address can point to a tag, folder or UDT. When reading a folder or UDT, all child tags, folders, and UDTs are also read. Examples are provided below.

```
// Read a line and all child data
[default]/site/are/line
```

```
// Read a tag
[default]/mytag

// Read the entire tag provider
[default]
```

Type: Path

Recursively reads the Tag Provider namespace at the specified address and returns all tags, folders, and UDT instances.

Type: UDT Instances

Recursively reads the Tag Provider namespace at the specified address and returns a list of UDT instances according to the **UDT Inclusion Filter**.

UDT Include Filter

Specifies the User Defined Types to match when searching the specified address. The input always produces a list of UDT instances. If empty, the list contains UDT instances of all types.

Output Settings

Outputs can write to or create tag providers, tags, folders, and UDTs (both definitions and instances). Outputs automatically detect changes to a Tag Provider and reflect those changes on the next write.

Tag Address

The tag address to write to. This address must exist if Create is disabled, otherwise the write fails. Like inputs, the tag address must include the tag provider and full path.

Create

Enable this option to create tag providers, tags, folders, and UDTs if they don't exist. See below examples of write payloads and how these map to the Ignition Address space.

Do not enable Create to write to existing tag providers that the Intelligence Hub has not created and does not own. In some cases this can lead to a loss of tags or UDTs in the existing tag provider.

Assume all the examples write to [myTagProvider] with Create enabled. If myTagProvider doesn't already exist, it's created.

The following write payload will create [myTagProvider]/myTag1 as an integer and [myTagProvider]/myFolder/mySubFolder/myTag2 as a string.

```
{
  "myTag1": 23,
  "myFolder": {
    "mySubFolder": {
      "myTag2": "hello world"
    }
  }
}
```

The following payload creates a UDT definition named "injection" with temp and cycleCount attributes. It also creates folder hierarchy for line1/area1 as well as two UDT instances named machine1 and machine2.

```
{
  "line1": {
```

```
    "area1": {
      "machine1": {"_model": "injection", "temp":  100, "cycleCount": 1},
      "machine1": {"_model": "injection", "temp":  75, "cycleCount": 20}
    }
  }
}
```

The following payload created a single “injection” UDT definition, and a UDT instance named “injection1”.

```
{"_model": "injection", "temp":  100, "cycleCount": 1}
```

Installing Module | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/ignition_module/ignition_module_install/

Installing Module

The Ignition Module is installed and runs on the Ignition server. It connects to Ignition using the Ignition Module Java SDK, and hosts a secure websocket for HighByte to connect to. To install the module, follow the steps below.

The module has been tested with Ignition version 8.1.33+.

Installing the HighByte Ignition Module

1. Download the `HighByte-Intelligence-Hub-X.X.X.modl` from the portal
2. Login to Ignition and navigate to Config and Modules. Select `Install or Upgrade Module...` at the bottom of the page
3. Click `Choose File` and browse and select the .modl file
    - You'll need to accept the module EULA and the self-signed certificate used to sign the module
4. Once installed, you'll see `HighByte Intelligence Hub` in the main left hand menu
5. You can now connect to Ignition from HighByte using the default port

Configuring the Modules

Enabled

When enabled the websocket servers is active and listening on the port.

Port

The port that the websocket server listens on.

Change Password? / Password

The password used to authenticate clients. This blank by default. If used, the password must match the password used by in the HighByte connection settings.

Kafka | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/kafka/

Kafka

The Kafka Connection allows the Intelligence Hub to connect to Apache Kafka brokers for reading (input) and writing (output) of messages. Below are details on the connection, input, and output settings and supported features.

Connection Settings

Broker URIs

Specify one or more Kafka broker URIs to connect to. Example: `broker1:9092,broker2:9092'. The connection uses the first broker in the list that successfully connects.

Security Type

Security type used for Kafka client connection.

- **Plaintext**: No security is used.
- **SSL**: Enables SSL for the client connection.
  - **Truststore CA Certificate**: The CA certificate used for the client connection.
  - **Keystore Client Keypair**: The client keypair used to encrypt/decrypt the client communication.

Input Settings

Kafka inputs subscribe to topics and consume messages. The following settings are available:

Setting	Description
Topic	The topic to subscribe to.
Group	The consumer group ID to join.
Initial Offset	Where to start consuming if no offset exists (e.g., earliest, latest).
Key Format	The expected format for the message key (Any, JSON, String, Binary). See options below.
Value Format	The expected format for the message value (Any, JSON, String, Binary). See options below.
Max Messages	Maximum number of records per read.
Include Metadata	Include metadata such as topic, partition, offset, timestamp, headers, key.

Key/Value Format Options

Type	Description
Any	The payload can be JSON, String, or Binary
JSON	The payload is deserialized as JSON. Non-JSON payloads are ignored.
String (UTF-8)	The payload is treated as a string.
Binary	The payload is treated as a byte array.

Output Settings

Kafka outputs publish messages to topics. The following settings are available:

Setting	Description
Topic	The topic to publish messages to.
Key	The key to use for the message (optional).
Value	The value to publish (payload).

MCP Client | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/mcp_client/

MCP Client

The MCP Client connection enables interaction with an MCP Server.

Connection Settings

Endpoint

The URL of the MCP Server. This should include the port number and any additional path information required by the server.

Note: We recommend not including a trailing forward slash (/) at the end of the Endpoint.

Authentication Type

Specifies the authentication method for connecting to the MCP Server.

None

No authentication will be used.

Bearer Token

Simple header-based token authentication.

Setting	Description
Token	The token included in the header for authentication

Enable MCP Tools

Enables the proxying of MCP Tools from the configured MCP server through the HighByte MCP server. All tool names will be prefixed with `connection_connectionName_` to prevent tool name collisions.

Input Settings

Inputs are not supported.

Output Settings

Outputs are not supported.

Modbus TCP | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/modbus_tcp/

Modbus TCP

The Modbus TCP connection allows the Intelligence Hub to connect to Modbus devices over TCP/IP, supporting both reading and writing of registers and coils. Below are details on the connection, input, and output settings.

Connection Settings

Setting	Description
Host	The hostname or IP address of the Modbus device.
Port	The port number the Modbus device is accepting connections on.
Unit Id	Controls the Unit ID in the header of the Modbus request. Default is 0.
Connect Timeout	Maximum time to wait for a connection to be established (1-300 seconds, default 10 seconds).
Request Timeout	Maximum time to wait for the device to respond (100-60000 ms, default 100 ms).
Zero Based Address	Whether 1 should be subtracted from any input's address before being sent to the device. Defaults to enabled.
Modbus Byte Order	When enabled, uses big endian for 16-bit register values. Big endian treats the first byte received as the high byte in a register value. Disable for little endian sources. Default enabled.
First Word Low	When enabled, the first 16-bit register value of two consecutive registers is treated as the low word to make up a in a 32-bit value. When disabled, the first 16-bit register value is treated as the high word. The default is enabled.
First DWord Low	When enabled, the first two 16-bit register values of four consecutive registers are treated as the low double word to make up a in a 64-bit value. When disabled, the first two 16-bit register values are treated as the high double word. The default is enabled.

Input Settings

Setting	Description
Start Address	The starting Modbus address to read. Valid start addresses: 000001 (output coil), 100001 (input coil), 300001 (input register), 400001 (holding register).
Count	The number of consecutive addresses to read, beginning with the start address. f the count is 1, the value will be provided as a scalar, otherwise the value will be provided as an array of size *Count*
Unit Id	(Optional) Overrides the connection-level Unit Id. Default -1 uses the connection setting.
Bit Length	(Holding registers only) Specifies the length: 16, 32, or 64.
Treat as Signed	(Holding registers only) Specifies if the value is signed or unsigned.

Output Settings

Setting	Description
Start Address	The starting Modbus address to write to. Same valid start addresses as input. Valid start addresses included 000001 (output coil), 100001 (input coil), 300001 (input register) and 400001 (holding register). For holding registers, Bit Length and Treat as Signed fields are also available.
Count	The number of consecutive addresses to write, beginning with the start address. If the count is 1, the value written is a scalar, otherwise the value is an array of size *Count*.
Unit Id	(Optional) Overrides the connection-level Unit Id. Default -1 uses the connection setting.
Bit Length	(Holding registers only) Specifies the length: 16, 32, or 64.
Treat as Signed	(Holding registers only) Specifies if the value is signed or unsigned.

MQTT | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/mqtt/

MQTT

The MQTT connection allows the Intelligence Hub to connect to MQTT brokers for reading and writing messages. This page describes the connection, input, and output settings, and supported features.

Connection Settings

Setting	Description
Client ID	The ID to identify the connection with the broker. Leave blank to generate a random ID. Each client must have a unique ID.
Username and Password	The username and password required by the MQTT broker.
Connection Timeout	Maximum time to wait for a connection to respond. Valid range is 1–300 seconds.
Keep Alive	Maximum time to wait without sending a request to the broker. Valid range is 10–43200 seconds.
Request Timeout (ms)	How long to wait for a read or write to complete before failing.
Redundant Brokers	A list of brokers to connect to if the primary connection is fails. Each broker in the list must contain the protocol, host, and port in the following format: tcp://127.0.0.1:8884 or ssl://127.0.0.1:8885. In the event that the primary connection fails the connection attempts the next broker in the list, and continues in round robin fashion until a connection is made. Once a connection is established the connector will continue to use the connection until it fails or the connection is re-saved.
Use SSL	Enable for MQTT brokers that require transport layer security. See additional SSL options below.
Input Discovery Topic	The topic the connection listens on to auto-discover topics. Leave blank to disable auto-discovery. When left blank, auto-discovery is disabled. When enabled, the connection subscribes on startup to the topic and the Browse Input feature shows all topics that have published data.
Clean Session	When enabled, the connection is not persisted. Events sent while not connected will not be received. When disabled, the connection is persisted using the provided Client ID. Use a unique Client ID and set QoS for inputs to 2 for reliable delivery.
Enable MCP Tools	When enabled, two MCP tools will be enabled for the connection: Browse and Read.

SSL Options:

- **Ignore Self-Signed Certificate**: Accepts any certificate returned from the broker. Use only for testing.
- **CA Certificate**: The MQTT Broker certificate file (e.g., `groupCA.pem`) imported into the certificate store.
- **Client Certificate**: The public certificate for the client authentication (e.g., `mycert.pem`).
- **Client Key**: The private key for the client authentication (e.g., `mycert.key.pem`).

Input Settings

Setting	Description
Input Type	Controls what is read. Last Publish returns only the last message published on a topic. Branch requires the topic to include a wildcard and returns all the currently known topics and values that meet the topic filter. See the Branch read example below. Finally, Browse returns a list of all topics which fall under a provided topic.
Topic	The topic to subscribe to for MQTT messages.
Include Topic	Specifies if the MQTT topic should be included in the read data. When enabled, the topic is included at the root of the payload. See example below.
Payload Type	The type of MQTT payload to process. See options below.

Payload Type Options

Type	Description
Any	The payload can be JSON, String, or Binary
JSON	The payload is deserialized as JSON. Non-JSON payloads ignored.
String (UTF-8)	The payload is treated as a string.
Binary	The payload is treated as a byte array.

Branch Read Example

Assume the broker has two topics, machine1Pressure with a value of 100, and machine2Pressure with a value of 200. A Branch read with a topic of # returns the following.

```
{
  "machine1Pressure": 100,
  "machine2Pressure": 200
}
```

Include Topic Example

```
{
  "_topic": "deviceid",
  "value": {
    "deviceid": "press1",
    "other": 10
  }
}
```

Output Settings

Setting	Description
Topic	The topic to publish MQTT messages to. Topics can be static or dynamic. Use `#` to include the source input or instance name, or `{{this.attribute}}` to include data from the payload.
QoS	Specifies the quality of service for publishing the output.
Named Root	Controls how the instance name appears in the output value. When enabled, output values are nested under the instance name.

Setting	Description
Retain	Specifies whether the MQTT broker should cache the last value sent.
Breakup Arrays	When enabled, array data is sent as individual messages for each element. Otherwise, the entire array is sent in one publish.
Template	If blank, the default JSON output format is used. See Output Templates for details.

AWS IoT Core

When using SSL, specifically with AWS IoT Core, AWS IoT Greengrass, or any broker that has a self-signed certificate, you will need to import the certificate into the HighByte Intelligence Hub certificate store. See the "Managing Certificates" section on how to import the certificates. You will need to import the self-signed CA certificate. You may also need to add the Client Certificate (as a certificate) and Client Key (as a key) if the broker is using certificate-based authentication.

MCP Tools

The default MCP Tools for an MQTT connection are detailed below. Note that tool calls will not cause the connection to connect or subscribe. The connection must be in use or the Input Discovery Topic setting must be used to ensure the connection and subscriptions are setup. Otherwise, there will be no topics.

Browse

Browse the topics that the MQTT connection has received data for.

Argument	Description
topic	The wildcard topic to browse. For example # returns all topics, or path/to/children/# returns all child topics of the path. The topic must contain a wildcard, otherwise it returns the original topic.

Returns an array all topics that meet the filter.


```
[
  "factory/line1/motor/temperature",
  "factory/line1/motor/speed",
  "factory/line1/sensor/vibration"
]
```


Read

Return the latest value for one or more topics.

Argument	Description
topics []	A string array of all the topics to read.

Assume the read it for ["topic1/child", "topic2"]. Below is the read result.


```
[
  {
    "_topic": "topic1/child",
    "value": "test"
  },
  {
    "_topic": "topic2",
    "value": 99
```

```
    }
]
```

OPC UA TCP | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/opc_ua_tcp/

OPC UA TCP

The OPC UA TCP Connection connects to an OPC UA server and supports browsing, reading and writing. The connection supports built-in and complex UA types, Alarms & Events, and Methods. Below are details on the connection, input, and output settings and supported features.

Connection Settings

Host, Port and Path

These settings control the IP address or hostname for the OPC UA Server, it's port, and the path. The path is informational only and not used as part of the connection.

Mode

The connection supports two modes; Subscribe or Poll.

Subscribe is the default mode, and recommended for most cases. In this mode, a single subscription is created to monitor data changes for all tags. The server sends the data changes on each Subscription Rate interval. The changes are cached locally in the connection, and reads pull from this cache. This mode works with both polled and event based triggers. Polled triggers read from the cache, and event triggers are triggered any time there is a data change for a tag.

In Polled mode, all reads are performed on-demand, meaning an explicit read is sent to the server asking for the current tag values. Depending on the server, this can make it harder for the server to optimize communications to the device, and generally is not recommended. This mode also does not work with event based triggers.

Subscription Rate

This setting only applies for Subscribe mode, and controls how often the subscription reports data changes for tags. This is also the default tag scan rate used by the OPC UA server to monitor tags. Data is received from the OPC UA server no faster than this rate.

Security

Level of security required by the OPC UA Sever. Valid options are None, Basic256Sha256-Sign and Basic256Sha256-SignEncrypt.

Security Policy: None

The Intelligence Hub supports the ability to turn off security as allowed by the OPC UA specification. In this mode, all communication is no encrypted (public) and not signed (unverified).

Security Policy: Basic256Sha256-Sign and Basic256Sha256-SignEncrypt

When security is enabled for the connection (Sign or SignEncrypt), the Intelligence Hub behaves as follows:

1. Automatically creates an application instance certificate the first time it is run (named `app-certificate-private-key`).
2. Connects to servers using asymmetric security algorithms encryption to securely exchange public keys and create a secure connection.
3. Signs or Signs and Encrypts all requests before sending it to the server.

4. Decrypts (if applicable) and verifies the signature of all responses coming from the server.

The first time the Intelligence Hub makes a secure connection to any OPC UA server, you will need to Trust the Intelligence Hub application certificate where the server is installed. Consult your OPC UA server documentation for instructions.

Authentication Type

The level of authentication required by the OPC UA Server. Valid options are Anonymous and Username-Basic256.

Setting	Description
Anonymous	No user credentials are passed as part of the connection.
Username_Basic256	Username and password are required and sent during the connection using Basic256 encryption defined by the OPC UA specification.

We recommend using a minimum of Basic256Sha256–Sign along with username/password authentication for all OPC UA communications.

Setting	Description
Client Certificate	Select the certificate used for secure communication with the OPC UA server. By default, an auto-generated certificate name `app-certificate-private-key` is used.
Connection Timeout	Maximum time to wait for a connection to be established. Valid range is 1–60 seconds.
Request Timeout	Maximum time to wait for a connection to respond. Valid range is 100–60000 milliseconds.
Max Items Per Read	Specifies the number of items that can be read each attempt. Increasing this value can add performance improvements to connections that have to read hundreds or thousands of items each time. However, increasing this value can also introduce stress onto the OPC UA server.
Max Items Per Write	Specifies the number of items that can be written each attempt. See performance considerations for the read setting above.
Session Name	Specifies the session name that is supplied to the server during session creation. When left blank, the product URN is used.
Browse Identifier Mode	Controls the identifier mode used when importing inputs/output during browse. See more: OPC UA Identifiers
Enable MCP Tools	When enabled, MCP tools will be enabled for the connection: Browse and Read.

OPC UA Identifier Mode

Input and Outputs can be configured to work in two Identifier Modes.

Node Identifier (NodeId)

Node identifiers (NodeIds) can be used to directly to access data, events, or methods in the OPC UA server. This is the preferred Identifier Mode unless the OPC UA server's NodeIds can change over time.

Setting	Description
Namespace Index	Numeric index of the namespace the identifier belongs to.
Identifier Type	Specifies the type of data used in the identifier field. This can be set to String or Numeric. This is not the data type of the value associated with this OPC UA input.
Identifier	Specifies the identifier used to reference the OPC UA input.

Path Identifier (Path)

A Path Identifier uses a Relative Path to locate a node in the OPC UA server namespace. A relative path consists of a list of uri:name tuples. All paths are relative to the **ObjectsFolder** node. At runtime the connection requests the server translate the browse path to current Node Id. This mode should be used when the OPC UA server NodeIds change over time because there is additional cost in translating paths to node ids at runtime.

This identifier mode is supported for Tag, Collection, and Branch Inputs as well as Outputs.

Setting	Description
Relative Path URI	Namespace URI of the relative path segment.
Relative Path Name	Name of the relative path segment.

Input Settings

Inputs have a number of types to read different data from the UA server. The types are summarized in the table. Some input types can be used in polled or event based data flows, and others can only be used in polled data flows.

The settings and details for each type are described below.

Type	Description	Data Flow Support
Tag	Reads the value for a single OPC UA tag/node.	Polled & Event
Collection	Reads a collection of tags that can exist in any part of the OPC UA address space.	Polled Only
Branch	Browses and reads all child nodes of the specified node in the OPC UA address space.	Polled Only
Method	Calls a Method using optional input parameters and optionally receive method result data from the OPC UA server.	Polled Only
Event	Subscribes to a node in the UA server and to receives alarms and events.	Polled & Event

Inputs support all built-in UA types, as well as custom structures and enumerated types (i.e. ExtensionObject in the UA specification).

For ExtensionObject types the UA server must include the schema definition in a DataType Dictionary in accordance to the OPC UA 1.03 specification or older. DataTypeDefinitions (1.04+) are not yet supported.

Type: Tag

A Tag input is used to read a single tag.

Setting	Description
Identifier Mode	Supports NodeId and Path.
Identifier	NodeId or Path for the tag.
Sampling Interval	Specifies the rate that the input should be monitored (read) for change. By default, this value is zero, which uses the subscription rate set on the connection. Set this value to something slower than the connection's subscription rate to sample less frequently (down sample). Setting this value to something faster than the connection's subscription rate will be clamped to the subscription rate (over sampling is not supported).
Include Metadata	When enabled the OPC UA input includes metadata in the read. The metadata includes the timestamp, quality, and tag name of the input.

Type: Collection

A Collection is a read of one or more tags that can exist anywhere in the OPC UA address space. The result is returned as a complex object containing all the tag names and their values.

Each tag in the collection has the following settings. Use the `Find` button to browse for tags to add to the collection.

Setting	Description
Name	The name of the tag in the collection. When the collection is read the resulting read is in `name: <value>` format.
Tag	Identifier Mode and Identifier are the same as the Tag type settings.

The collection has the following additional settings.

Type	Description
Ignore Bad Quality	When enabled, if some tags in the collection are bad quality, those tags are omitted from the result. When disabled, if any tag in the collection is bad the read fails.

Type: Branch

A Branch input is a read on a branch node in the OPC UA server that returns the nodes children.

Below is an example of an OPC UA hierarchy and the result Branch read value.

```
// OPC UA Branch structure
- rootBranch
    - childBranch1
        - tag1
        - tag2
    - childBranch2
        - tag1

// The result of the read in JSON
"rootBranch": {
    "childBranch1": {
        "tag1": 100,
        "tag2": "hello world",
    },
    "childBranch2": {
        "tag1": 1.2
    }
}
```

Setting	Description
Branch	The root node for the branch. Identifier Mode and Identifier are the same as the Tag type settings.
Max Branch Depth	Limits the depth of the resulting read. For example, if set to 2 and a branch has 4 levels of hierarchy, only tags in the first 2 levels of hierarchy are read
Cache Interval	Controls how often the input browses the OPC branch to detect changes, like new tags being added or removed. On the first branch read the input browses up to Max Branch Depth and caches the tags and addresses. Subsequent reads use the cache to issue reads for efficiency. After the cache interval expires, the cache is cleared and the input re-browses, picking up any changes made to the branch. Changing the input configuration or saving the connection will also clear the cache.
Ignore Bad Quality	When enabled, if some tags in the branch are bad, those tags are omitted from the result. When disabled, if any tag in the branch is bad the read fails.

Type: Method

Method inputs execute an OPC UA Method in the server. Methods can take arguments and return values. It is recommended to use the browse feature to find and configure method inputs.

Setting	Description
Parent Object	The parent node that has the method. NamespaceIndex, Identifier Type, and Identifier are the same as the Tag type settings. Path Identifier Mode is not supported.
Method	The node identifier for the method. NamespaceIndex, Identifier Type, and Identifier are the same as the Tag type settings. Path Identifier Mode is not supported.
Input Arguments	Defines one or more input arguments to pass to the method. Each argument has a value and data type. The value can be a static value or a reference. If the data type is set to Auto, the value is converted to the data type required by the method. Otherwise, set the data type to the method input argument data type. The name and description fields are optional, and are not sent to the OPC server.
Output Arguments	OPC UA methods can return one of more output arguments. This setting allows you to assign each argument a name.

Method Output Arguments Example

If a method returns two values and no output arguments are defined the read result looks as follows.

```
{
  "outputArguments": {
    "output0": 0,
    "output1": 0
  }
}
```

If two output arguments are defined “myArg1” and “myArg2” the result looks as follows.

```
{
  "outputArguments": {
    "myArg1": 0,
    "myArg2": 0
  }
}
```

Type: Event

Event inputs subscribe to Alarms and Events for a node, and allow configuration of what attributes are returned as part of the event.

Note the input does not request a refresh on the first read, and will not return the current state of conditions. The first read subscribes to the node with the event filter, and only returns a result when a new event is generated in the server.

The `Find` feature for the Attribute Filter setting browses the OPC UA server for all event and condition types, and available attributes.

Setting	Description
Tag/Node	The node to monitor for alarms and events. For example, this could be the Server node, in which case all events are subscribed to for the server. NamespaceIndex, Identifier Type, and Identifier are the same as the Tag type settings. Path Identifier Mode is not supported.

Setting	Description
Attribute Filter	The list of attributes to subscribe to for the event. This directly controls what data the event can return. If the event contains the attribute, it's included in the payload, otherwise it's omitted.

Attribute Filter Examples

By default, when there are no attributes in the filter, the input subscribes to the base EventId, SourceName, Severity, Time, Message, ConditionName, and Retain attributes. EventId is converted to a GUID if possible. Below is an example read result from the default filter.

```
{
  "eventId": "18a12e4c-40f1-9f46-924b-d62bd3deab21",
  "sourceName": "ExclusiveLevelAlarmTrigger",
  "severity": 800,
  "time": "2024-03-15T17:51:09.448Z",
  "message": "Exclusive level alarm active in LowLow",
  "conditionName": "ExclusiveLevelAlarm",
  "retain": true
}
```

Now assume the `Find` feature is used to find Event/Condition types and select only the EventId, SourceName, and MyCustomAttribute attributes for the filter. The read result looks follows. Note MyCustomAttribute is not in the event data, so it's ignored.

```
{
  "eventId": "18a12e4c-40f1-9f46-924b-d62bd3deab21",
  "sourceName": "ExclusiveLevelAlarmTrigger"
}
```

Output Settings

OPC UA outputs support writing to single tag or a branch. When writing to a branch, the out tries to match the write data to the OPC UA branch address space. Below is an example of how this mapping works.

Outputs do not support writing to complex structures at this time.

```
// Example payload being written to OPC UA Output
{
    "branch1": {
        "tag1": 1,
        "branch2": {
            "tag2": 2,
            "branch3": {
                "tag3": 3
            }
        }
    }
}

// Example OPC UA branch structure. In this example the payload's branch1.branch2.tag value
// written. tag1 and tag3 do not map and are ignored.
- branch1
    - branch2
```

```
    - tag2
    - branch3
```

Identifier Mode

Identifier mode to use for the output. Supports NodeId and Path.

Data Type

The data type used for the write. OPC UA is generally very strict on matching the data type of the server and this option provides control over this from the client side. The following options exists:

- **Auto**: The client will browse or read the UA Server tags data type before the write and attempt to convert the write value to the UA Server required value. This option is often used when writing complex data to a branch
- **Implicit**: The client will use the data type of the value being written.
- **Boolean, String, etc**: Use this data type for the write.

If the output cannot convert that write type to the type specified by this setting, the write will fail and output an error to the logs.

Max Branch Depth

When writing to a branch, the output first does a browse to get the tags for the branch. This setting controls how deep the browse will traverse. From the above example, if the Max Branch Depth was set to 1, tag2 would not be browsed and written to, and would instead be ignored.

Cache Interval

Like the branch reads setting, this controls how often the browse cache is refreshed to detect changes in the branch.

Named Root

When disabled, the payload is mapped to the OPC UA server's branch structure as expected. When enabled, this setting allows writing multiple Instances to a single branch, where the root branch has a child branches with each Instance name. This allows a single pipeline to send multiple instances to the same output. For example, assume the OPC UA root branch looks as follows. Enabling this setting will cause the write to map to `rootBranch.instance1` versus just rootBranch.

```
- rootBranch
    - instance1
        - temp
    - instance2
        - temp
```

MCP Tools

The default MCP Tools for an OPC UA connection are detailed below.

Browse

Browse nodeid hierarchy in the OPC UA address space.

Setting	Description
nodeid	The OPC UA nodeid to start the browse. Leave this blank to browse the root. (ex. ns=2;s=tagFolder)

Returns an array of all child nodes; including the display name, nodeid, and if the node has children to browse further.

```
[
  {
    "name": "Server",
    "nodeid": "ns=0;i=2253",
    "hasChildren": true
  },
  {
    "name": "granulator",
    "nodeid": "ns=2;s=pressure",
    "hasChildren": false
  }
]
```


Read

Reads the value of one or more nodeids.

Argument	Description
nodeids []	An array of string nodeids to read.

Assume the input is ['ns=2;s=FertilizerDemo.Line1.granulator.rpm', 'ns=2;s=FertilizerDemo.Line1.granulator.status']. The result is as follows.


```
[
  {
  "nodeid": "ns=2;s=FertilizerDemo.Line1.granulator.rpm",
  "value": 0,
  "quality": "Good",
  "timestamp": "2025-10-10T16:34:41.626Z"
  },
  {
  "nodeid": "ns=2;s=FertilizerDemo.Line1.granulator.status",
  "value": false,
  "quality": "Good",
  "timestamp": "2025-10-10T16:34:41.626Z"
  }
]
```

OPC-UA CSV Import | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/opc_ua_tcp/csv_import/

OPC-UA CSV Import

OPC UA supports importing inputs from a CSV file. This section covers the import format. Below the columns for different inputs types as well as the common columns are outlined. A single CSV file can include all types by including all columns and leaving columns empty for a given row if it does not apply.

Basic Input Columns

Column	Description
`name`	The name of the input.
`cacheLifetime.enabled`	(optional) True if input caching is enabled for the input.
`cacheLifetime.interval.duration`	(optional) The duration of the cache (ex 1 minute)
`cacheLifetime.interval.units`	(optional) The units for the duration. Milliseconds, Minutes, Hours.
`template.name`	(optional) Name of the templated input.
`template.type`	(optional) Type of template; Static or Dynamic.
`template.reference`	(optional) For Dynamic types only, the reference (ex. {{Connection.CSV.headers}})
`template.params.paramName`	(optional) Template parameters, where paramName is the template parameter name and the value is the range (ex. 1-10). Add a column for each parameter.
`qualifier.type`	The type of OPC UA input. Valid types are Tag, Collection, or Method. This controls the remaining qualifier columns

Tag Columns

Column	Description
`qualifier.samplingInterval.duration`	(optional) Override the default sampling interval for the tag.
`qualifier.samplingInterval.units`	(optional) Units for the duration.
`qualifier.namespaceIndex`	The namespace index for the tag (ex. 2)
`qualifier.identifierType`	The type of identifier for the tag. String and GUID are the most common types.
`qualifier.identifier`	The identifier for the tag.
`qualifier.isComplex`	(optional) Enable to include meta data with the tag.

Collection Columns

Collections are expressed as multiple rows, where the first row includes the Basic Input Columns and the first tag in the Collection. Proceeding rows leave the Basic Input Columns empty and fill in the collection columns with the remaining tags in the collection

Column	Description
`qualifier.collection.name`	The name of the tag in the collection.
`qualifier.collection.namespaceIndex`	The namespace index for the tag (ex. 2)
`qualifier.collection.identifierType`	The type of identifier for the tag. String and GUID are the most common types.
`qualifier.collection.identifier`	The identifier for the tag.

Branch Columns

Branches are much like single tags, but the reference is to an OPC UA branch.

Column	Description
`qualifier.namespaceIndex`	The namespace index for the tag (ex. 2)
`qualifier.identifierType`	The type of identifier for the tag. String and GUID are the most common types.
`qualifier.identifier`	The identifier for the tag.
`qualifier.maxDepth`	(optional) Max depth to browse and read. 0 means one level, 1 means two levels, etc.
`qualifier.cacheInterval.duration`	(optional) How often to refresh the browse hierarchy.
`qualifier.cacheInterval.units`	(optional) The units for how often to refresh the browse hierarchy.
`qualifier.ignoreBadQuality`	(optional) Enable to ignore bad quality values.

Method Columns

Column	Description
`qualifier.namespaceIndex`	The namespace index for the parent object that has the method (ex. 2)
`qualifier.identifierType`	The type of identifier for the parent object.
`qualifier.identifier`	The identifier for the parent object.
`qualifier.method.namespaceIndex`	The namespace index for the method.
`qualifier.method.identifierType`	The identifier type of the method.
`qualifier.method.identifier`	The identifier for the method.

Troubleshooting | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/opc_ua_tcp/troubleshoot/

Troubleshooting

Error Messages

Bad_CertificateUriInvalid, value=0x80170000, The URI specified in the ApplicationDescription does not match the URI in the Certificate

Description

OPC UA requires the server and client to provide a certificate at the time of connection, regardless of whether security is used. The Certificate contains a SubjectAlternative name with a value of URL=urn:hostname:appname. This URL must match the URL defined in the ApplicationDescription which is also passed during the connection. Both sides must verify the certificate and ApplicationDescription match. If they don't match the connection is closed with the above error.

Solution

1. If HighByte is running inside of docker, make sure the docker configuration sets the hostname. Otherwise the container gets a random hostname each time it's restarted. The certificate is cached for each run and will contain the hostname of the first run. To regenerate the certificate, delete the 'app-certificate-private-key' in the certificates list.
2. Verify the URL for the client and server certificates match the hostnames.
3. Verify there are no special characters (ex. spaces) in the URL. These must be URL encoded in the certificate (i.e. space is %20)

REST Client | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/rest_client/

REST Client

The REST Client connection allows for sending and receiving data to/from HTTP endpoints. The connection is flexible to communicate with the many different types of REST endpoints, both in authentication schemes and data formats.

Connection Settings

Base URL

The base URL for the request (e.g., https://myurl.com)

Note: We recommend trailing forward slash is not included in the Base URL.

Headers

Name, Value header pairs to be passed on each request.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Note: Depending on the authentication type, some headers may be overwritten. Eg, if "Basic Auth" is chosen, the "Authentication" header will be automatically set.

Authentication Type

Authentication type used for REST client connection.

None

No authentication used.

Basic Auth

Basic authentication included in the header. Set the username and password.

OAuth 2.0

Client Credentials and Password grant types are supported. Credentials can be sent in the header or the body. If an input or output receives a 401 (unauthorized), the request is attempted a second time allowing the token to be refreshed without data loss.

Setting	Description
Grant Type	Client Credentials or Password
Login URL	The login URL of the OAuth provider.
Username	The user's username. Only used with Password grant type.
Password	The user's password. Only used with Password grant type.
Client ID	The client id.
Client Secret	The client secret.

Setting	Description
Scope	Optional scope of the client.
Audience	Optional audience.
Resources	Optional resources.
Client Authentication	Send the credentials in the header or the body.

Client Credentials

The Client Credentials grant type is used by clients to obtain an access token outside of the context of a user.

Password

The Password grant type is a legacy way to exchange a user's credentials for an access token.

Bearer Token

Simple authentication using a header based token.

Setting	Description
Token	The token included in the header for login

Ignore Self-Signed Certificate

Used to ignore validation of the certificate if https:// is used. This is useful for testing but not recommended in production.

Input Settings

Inputs can send requests to an HTTP endpoint and make the response available as data in the server.

Endpoint URL

The endpoint to send the request to (e.g., `/routes/myendpoint`). A leading / is not required, but recommended.

Content Type

The content type of the incoming request: JSON or XML are supported.

HTTP Method

The method to use: GET or POST.

Headers

Name, Value header pairs to be passed on each request.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

These are added to the connection level header.

Note: Depending on the authentication type, some headers may be overwritten. Eg, if "Basic Auth" is chosen, the "Authentication" header will be automatically set.

Request Body

The body of the POST request. This can be any text including JSON.

Raw Response

Enable this to include the returned value, status code, and headers in the response. See the example read result below.

When enabled, non-200 responses do not produce a read error, and instead return the status code as shown below. The only exception to this are 401 error codes. These indicate an authentication issue, and produce a read error.


```
{
  "value": {
    "tag1": 100
  },
    "_statusCode": 200,
    "_headers": {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Headers": "origin, content-type, accept, authorization, x-key-id,
    "Access-Control-Allow-Credentials": "true",
    "Access-Control-Allow-Methods": "GET, POST, PUT, DELETE, OPTIONS",
    "Content-Type": "application/json",
    "Transfer-Encoding": "chunked"
    }
}
```


Output Settings

Outputs can send data to an HTTP endpoint.

Endpoint URL

The endpoint to send the request to (e.g., `/routes/myendpoint`). Leading / is not required but recommended.

Headers

Name, Value header pairs to be passed on each request.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

These are added to the connection level header.

Note: Depending on the authentication type, some headers may be overwritten. Eg, if “Basic Auth” is chosen, the “Authentication” header will be automatically set.

HTTP Method

The method to use: `POST` or `PUT`.

Body Type

Controls how the data is sent as part of the HTTP request.

- **None** No data is sent. This is useful for triggering external webhooks.
- **Form Data** Data is encoded as a multipart form.
- **Form Urlencoded** Data is URL encoded and sent in the request body.
- **Template** Enter a custom template in the template field. If empty, back to default JSON encoding. See Output Templates for details on the template syntax.
- **Binary** File is sent in binary format. Complex data falls back to default JSON encoding.

Snowflake SQL | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/snowflake_sql/

Snowflake SQL

The Snowflake SQL connection connects to Snowflake using JDBC driver. The connection supports reading data from Snowflake using SQL queries.

Connection Settings

Account Identifier

The identifier for the Snowflake account, which can be found when logging into the Snowflake portal. See the Snowflake documentation for more details. An example format is ZHX29753.

https://docs.snowflake.com/en/user-guide/admin-account-identifier

User

The user account in Snowflake to authenticate with.

Role

The role of this user account. The role must have the ability to create tables if create is used in the Output settings.

Private Key

The private key used to authenticate the user. See the Snowflake documentation for generating private key for an account. The private key must be imported into the Intelligence Hub certificate store (See Certificates) and referenced here.

https://docs.snowflake.com/en/user-guide/key-pair-auth

Database

The Snowflake database to write to.

Schema

The default schema to use. Defaults to `Public`.

Warehouse

The warehouse to use for any processing of data.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

See JDBC Driver Settings for input settings.

Errors with JVM Version

Depending on your JVM, you may see an error similar to the following when reading a Snowflake SQL input.

```
JDBC driver internal error: exception creating result java.lang.NoClassDefFoundError: Could
```

This is caused by an issue with Arrow, an open source Apache project used by the Snowflake JDBC driver to provide a more efficient binary format for exchanging data with Snowflake. This issue exists on JVM versions 16 and 17. To get around this issue there are a few options.

1. Upgrade your JVM to the latest v21
2. Add a JVM_OPTION via the command line or environments variable to find/load the dependency
    1. Command line: `--add-opens=java.base/java.nio=ALL-UNNAMED`
    2. ENV variable: `_JAVA_OPTIONS="--add-opens=java.base/java.nio=ALL-UNNAMED"`
3. Once the connection is made in HighByte, run the following SQL query on the input. Note this needs to be run any time the connection reconnects. This command skips the Arrow binary format and exchanges data in JSON.
    1. `ALTER SESSION SET JDBC_QUERY_RESULT_FORMAT='JSON'`

Output Settings

Table

The name of the table to write to.

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are *Off*, *Create*, and *Create*.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. All tables are created with an auto incrementing _id column used as the primary key. Snowflake uses a sequence to generate the values for an auto-incremented column. Sequences have limitations; see Sequence Semantics for more details.

Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 2048 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR(2048)
DateTime	DATETIME
UInt8	NUMBER
Int8	NUMBER
UInt16	NUMBER
Int16	NUMBER
UInt32	NUMBER
Int32	NUMBER
UInt64	NUMBER
Int64	NUMBER
Real32	FLOAT
Real64	FLOAT

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when **Create Table** is set to Off. Disable this refresh by setting the time interval to 0.

Case Sensitive

By default, Snowflake is case insensitive, and all tables and columns are upper case. This is the recommended approach. When writing to a table that is case sensitive, the table name and columns must be wrapped in quotes. Enabling this option wraps all names in quotes.

Snowflake Streaming | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/snowflake_streaming/

Snowflake Streaming

The Snowflake Streaming connection connects to Snowflake using the Snowpipe Streaming API. The connection also uses the Snowflake JDBC driver to search for and create tables.

The Intelligence Hub connects outbound on port 443 to Snowflake (example: https://accountid.snowflakecomputing.com).

Connection Settings

Account Identifier

The identifier for the Snowflake account, which can be found when logging into the Snowflake portal. See the Snowflake documentation for more details. An example format is ZHX29753.us-east-1.

https://docs.snowflake.com/en/user-guide/admin-account-identifier

User

The user account in Snowflake to authenticate with.

Role

The role of this user account. The role must have the ability to create tables if create is used in the Output settings.

Private Key

The private key used to authenticate the user. See the Snowflake documentation for generating private key for an account. The private key must be imported into the Intelligence Hub certificate store (See Certificates) and referenced here.

https://docs.snowflake.com/en/user-guide/key-pair-auth

Database

The Snowflake database to write to.

Schema

The default schema to use. Defaults to `Public`.

Warehouse

The warehouse to use for any processing of data.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Output Settings

Outputs can create and stream data to tables in the Snowflake database.

Table

The name of the table.

Create Table

Controls how the table definition is managed. Options are *Off*, *Create*, and *Create & Update*.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

Case Sensitive

By default, Snowflake is case insensitive, and all tables and columns are upper case. This is the recommended approach. When writing to a table that is case sensitive, the table name and columns must be wrapped in quotes. Enabling this option wraps all names in quotes.

Output Example

Below is an example of how an output payload maps to a Snowflake table. If the table is already created, payload attributes are mapped to existing column names. If the columns don't exist the data is omitted.

Example payload:

```
{
  "col1": 123,
  "col2": 1.23,
  "col3": "123",
  "col4": true,
  "col5": "2023-11-02T14:31:55Z",   // shown as string, but assume this is a DateTime type
  "col6": {
    "col6_attr1": 123,
    "col6_attr2": [1,2,3]
  }
}
```

Snowflake table:

Column Name	Data Type	Notes
COL1	NUMBER	
COL2	FLOAT	
COL3	NVARCHAR(1024)	
COL4	BOOLEAN	
COL5	DATETIME	
COL6	NVARCHAR(16777216)	Stored as JSON {"col6_attr1": 123, "col6_attr2": [1,2,3]}

Sparkplug | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/sparkplug/

Sparkplug

The Sparkplug Connection connects to an MQTT Broker and supports reading and writing data in Sparkplug format. The connection can be used to read data from one or more Group/Edge Node/Device, or can be setup to publish as a Group/Edge Node/Device.

Currently SparkplugB is supported. Below are details on the connection, input, and output settings and supported features.

Connection Settings

Host

The IP or hostname of the MQTT broker.

Port

The port of the MQTT broker.

Client ID

The ID to send to identify the connection with the broker. Leave blank to make this random. MQTT brokers require each client to have a unique ID and will disconnect clients with duplicate IDs.

Username and Password

The username and password required by the MQTT broker.

Connection Timeout

Maximum time to wait for a connection to respond. Valid range is 1–300 seconds.

Keep Alive

Maximum time to wait without sending a request to the broker. Valid range is 10-43200 seconds.

Request Timeout (ms)

Maximum time to wait for a subscription to arrive. Valid range is 100–60000 milliseconds.

Default Group Id / Edge Id

The default Group Id and Edge Id to use if the input or output settings don't override them.

If these are set the connection will setup as Last Will NDEATH message for the node on connect. The connection will also subscribe using the Group/Edge for NCMD messages to listen and respond to NBIRTH requests. If this isn't desired, leave these blank and set the Group/Edge in the Input and Output settings

Use SSL

Enable for MQTT brokers that require transport layer security. information on MQTT security.

Ignore Self-Signed Certificate

When enabled, any certificate returned from the broker will be accepted. This setting should only be used for testing connectivity, but is not recommended for production.

CA Certificate

The name of the trusted certificate file used to identify the broker as part of a secure TLS connection.

Client Certificate

Used if the broker broker requires client authentication using a certificate. This is the public certificate for the client (e.g., mycert.pem).

Client Key

Used if the broker requires client authentication. This is the private key for the client (e.g., mycert.key.pem).

Input Settings

Inputs can read a single metric on a group/edge node/device or the entire Sparkplug namespace. Inputs support reading quality (good/bad) but currently do not support reading Sparkplug properties.

Setting	Description
Group Id	Override the Default Group Id field set in the connection settings. This makes the input listen on topic `spBv1.0/<group_id>/#`. Group Id can be a wildcard '#' which means the input listens on all groups on the broker (i.e. `spBv1.0/#`). In this mode the input cannot request a birth certificate for groups/edge nodes because there is no way to discover them. Instead, it waits to receive a message from a new group/edge node and then immediately requests a birth. This can cause a delay in groups/edge nodes appearing in the read result.
Edge Node Id	Override the Default Edge Node Id field set in the connection settings. This makes the input listen on topic `spBv1.0/<group_id>/<msg_type>/<edge_id>/#`, gathering all data and devices for the edge node. Edge Node Id can be a wildcard '#' which means the input listens on a group for all edge nodes (i.e. `spBv1.0/<group_id>/<msg_type>/#`). In this mode the input cannot request a birth certificate for edge nodes because there is no way to discover them. Instead, it waits to receive a message from a new edge node and then immediately requests a birth. This can cause a delay in edge nodes appearing in the read result.
Device Id	Allows for filtering data from an edge node by device. A wildcard `#` is supported to get all devices for the edge node. Leave this blank to only collect metrics for the edge node.
Metric Name	Prides a way to filter the result to return only a given metric, or a branch of metrics. For example `Folder/MetricA` only returns MetricA, where `Folder/#` returns all metrics under Folder.
Message Type	DATA or CMD. DATA messages include the metric information only. CMD messages capture write commands issued to the `Group/Edge/Device/Metric`. Note CMD's can only be read once when they are used in a data flow. This way a single write isn't issued multiple times.
Include Metadata	When enabled, the Edge of Node, Group, and Device ID (if applicable) are included in the payload of the read. The value is included under a "value" attribute.
Include Properties	When enabled, properties belonging to the value are included in the payload. The value is included under a "value" attribute. This option is only available when the Message Type is set to DATA.

Output Settings

Outputs support writing as a single or many group/edge node/devices (using Dynamic Outputs). Outputs support writing metric values and a data quality property (good/bad) as well as responding to birth/death message requests.

Setting	Description
Group Id	Override the Default Group Id field set in the connection settings.
Edge Node Id	Override the Default Edge Node Id field set in the connection settings.
Device Id	The Id for the attached device the metric belongs to. Do not set the device id if the metric belongs to the EoN.
Metric Name	Optionally specify the metric name to write. Examples include: MetricA, Folder/MetricA. If you do not set a metric name, the metric name will be auto generated from the input or model instance name. This allows you to route many inputs and/or instances to a single output and simplifies configuration.
Message Type	DATA or CMD. DATA sends NDATA or DDATA messages. CMD is used to send write commands for one or more metrics.
Timestamp	If specified, the timestamp in the payload will be overwritten by the given timestamp. This field accepts both time since epoch in milliseconds or an ISO8601 date time string in UTC format.
Publish Type	Controls how the data is published, either in Tag or Template format. In Tag format data is flattened into metrics with a “/” delimiter. In Template mode metrics are converted into SPB templates, including hierarchy.

SQL | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/sql/

SQL

- JDBC Driver
- Microsoft SQL Server
- MySQL
- Oracle Database
- PostgreSQL
- SQLite

JDBC Driver | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/sql/jdbc_driver/

JDBC Driver

Connection Settings

The JDBC connection works with any 3rd-party JDBC drivers. To use this connection you must manually load the JDBC .jar file and dependencies into the runtime/libs directory.

JDBC Connection String

The full JDBC connection string required by the driver. Please consult your driver documentation.

Class Path

Class path of the JDBC driver. Please consult your driver documentation.

Username

Username for authentication.

Password

Password for authentication.

SQL Syntax

Controls the quoting and case sensitivity syntax used by the driver. Options include the syntax used by other common SQL drivers. If not set, SQL Server syntax is used.

Input Settings

Sends the query to the database and returns the generated result, if there is any.

Query

The full SQL query, which can include any SQL syntax including stored procedures.

Index

This setting is used to index the SQL query and only get new data on each query. When enabled, specifies a Column Name and default Value to reference within the query. This name/value pair is cached across successive product runs and allows the next query to pick up where it left off.

Note the Column Name must be the column name in the table that is being used for indexing. An example query might be SELECT * FROM SimpleMachineX WHERE id > {{index}} By enabling indexing and specifying the Column Name as id and the Value as zero, the first time the query is run {{index}} is set to zero and after the query {{index}} is updated to the latest value from id. The latest value is cached to disk. Successive queries will use the latest value cached and then update the cached value on success.

To reset the index, navigate to the “Index” page and delete the index value. Note that test reads on the input done from the UI will use the current index Value but will not update the index Value.

Column Name

Specifies the column name used to index. To update the index, the column name must be returned in the SQL query.

Value

Specifies the initial value used for the index.

Output Settings

Table

The name of the SQL table or stored procedure to write to. The table must already exist and have column names that match the model attribute names. Table names containing capital letters must be placed inside quotes (e.g., "MyTable").

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are Off, Create, and Create.

- When Off is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When Create is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When Create & Update is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. All tables are created with an auto incrementing _id column used as the primary key. Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 256 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR(2048)
DateTime	DATETIME
UInt8	CHAR
Int8	CHAR
UInt16	SMALLINT
Int16	SMALLINT

Write Data Type	SQL Data Type
UInt32	INTEGER
Int32	INTEGER
UInt64	BIGINT
Int64	BIGINT
Real32	FLOAT
Real64	FLOAT

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when Create Table is set to Off. Disable this refresh by setting the time interval to 0.

Keep Alive

Periodically perform an operation on the connection to prevent it from timing out.

Setting	***Description***
Enabled	Specifies whether keep alive is enabled for the connection.
Keep Alive Interval	How often an operation will be performed against the connection to avoid connection timeouts. The interval can be specified in milliseconds, seconds, minutes, hours, or days. Valid range is between 10 milliseconds and 36 days.

Microsoft SQL Server | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/sql/microsoft_sql_server/

Microsoft SQL Server

Connection Settings

Database

Name of the database to connect to.

Username

Username for authentication with the database.

Password

Password for authentication with the database.

Additional JDBC Options

Additional JDBC options to include in the JDBC connection string. This is MSSQL specific. Options are entered exactly as they would be if the JDBC connection string was created manually.

For example, if there are multiple instances running on the same server `instanceName` can be specified in this field. (eg. `instanceName=myInstanceName`)

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

Type

The type of input to use. The options are *Query* and *CDC*.

- When using a *Query* input, users can send a query to the corresponding database.
- When using a *CDC* input, users can track changes on a specified table

Query Settings

Sends the query to the database and returns the generated result, if there is any.

Query

The full SQL query, which can include any SQL syntax including stored procedures.

Index

This setting is used to index the SQL query and only get new data on each query. When enabled, specifies a Column Name and default Value to reference within the query. This name/value pair is cached across successive product runs and allows the next query to pick up where it left off.

Note the Column Name must be the column name in the table that is being used for indexing. An example query might be SELECT * FROM SimpleMachineX WHERE id > {{index}} By enabling indexing and specifying the Column Name as id and the Value as zero, the first time the query is run {{index}} is set to zero and after the query {{index}} is updated to the last value from id. The last value is cached to disk. Successive queries will use the last value cached and then update the cached value on success.

To reset the index, navigate to the "Index" page and delete the index value. Note that test reads on the input done from the UI will use the current index Value but will not update the index Value.

Column Name

Specifies the column name used to index. To update the index, the column name must be returned in the SQL query.

Value

Specifies the initial value used for the index.

CDC Settings

Generates a standardized value that represents a SQL change event.

```
{
    "table": {
        "schema": "dbo", // The schema name for the tracked table
        "name": "cdc_table" // The name of the tracked table
    },
    "operation": "Update", // Can be Insert, Update, or Delete
    "primaryKey": {
        // Contains each primary key that belongs to the row along with their corresponding
    },
    "before": {
        // The state of the row before the operation occurred
        // Is available for Update and Delete operations
    },
    "after": {
        // The state of the row after the operation occurred
        // Is available for Update and Insert operations
    }
}
```

Insert Operation

```
{
    "table": {
        "schema": "dbo",
        "name": "cdc_table"
    },
    "operation": "Insert",
    "primaryKey": {
        "machineId": 1
    },
    "after": {
        "machineId": 1,
        "speed": 0.532
    }
}
```

Update Operation

```
{
    "table": {
        "schema": "dbo",
        "name": "cdc_table"
    },
    "operation": "Update",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.532
    },
    "after": {
        "machineId": 1,
        "speed": 0.567
    }
}
```

Delete Operation

```
{
    "table": {
        "schema": "dbo",
        "name": "cdc_table"
    },
    "operation": "Delete",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.567
    }
}
```

Table Name

Name of the table to track changes for.

Transaction Count

The max number of transaction (LSN) changes that are retrieved per read. Each transaction (LSN) could have multiple changes/rows associated with it.

Output Settings

Table

The name of the SQL table or stored procedure to write to. The table must already exist and have column names that match the model attribute names. Table names containing capital letters must be placed inside quotes (e.g., “MyTable”).

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are **Off**, **Create**, and **Create & Update**.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. All tables are created with an auto incrementing _id column used as the primary key. Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 256 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR(2048)
DateTime	DATETIME
UInt8	CHAR
Int8	CHAR
UInt16	SMALLINT
Int16	SMALLINT
UInt32	INTEGER
Int32	INTEGER
UInt64	BIGINT
Int64	BIGINT
Real32	FLOAT
Real64	FLOAT

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when Create Table is set to Off. Disable this refresh by setting the time interval to 0.

MSSQL Windows Authentication

To configure the MSSQL connector to use Windows Authentication perform the following steps.

1. Set `integratedSecurity=true` in the Connection String
    - Found in the details pane of the MSSQL Connection in HighByte Intelligence Hub
    - Leave `User` and `Password` field blank
2. Stop HighByte Runtime
    - For Windows, execute `stop-windows.bat` file in HB runtime directory
3. Download Microsoft's JDBC package https://github.com/microsoft/mssql-jdbc/releases/tag/v9.2.1. This is a native Windows dependency required by the MSSQL JDBC driver to authenticate. Ensure you download the correct version for your MSSQL server.
    - Download `mssql-jdbc-9.2.1.jre11.jar` and `mssql-jdbc_auth.zip`
4. Extract file `mssql-jdbc_auth-9.2.1.x64.dll` found in `mssql-jdbc_auth.zip` at the path `\auth\x64\mssql-jdbc_auth-9.2.1.x64.dll`
5. Copy `mssql-jdbc_auth-9.2.1.x64.dll` to `HighByte-Intelligence-Hub-x.x.x\runtime` directory
6. Delete file mssql-jdbc-8.4.1.jre11.jar found in `HighByte-Intelligence-Hub-x.x.x\runtime\lib` directory
7. Copy mssql-jdbc-9.2.1.jre11.jar to `HighByte-Intelligence-Hub-x.x.x\runtime\lib` directory
8. Start HighByte Runtime
    - For Windows, execute start-windows.bat file in HB runtime directory

Considerations

1. Please ensure that the jar version and dll version match.
2. This will fix the `This driver is not configured for integrated authentication` error.
3. Ensure TCP/IP connections are enabled for the MSSQL Server.

SSL Connection Errors

When connecting to older SQL servers with older encryption, it's possible to see errors like this.

"The driver could not establish a secure connection to SQL Server by using Secure Sockets Layer (SSL) encryption. Error: "The server selected protocol version TLS10 is not accepted by client preferences [TLS13, TLS12]"

To get around this connection issue add `useSSL=false` to the JDBC Connection string.

MySQL | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/sql/mysql/

MySQL

Connection Settings

Database

Name of the database to connect to.

Username

Username for authentication with the database.

Password

Password for authentication with the database.

Additional JDBC Options

Additional JDBC options to include in the JDBC connection string. This is MySQL specific. Options are entered exactly as they would be if the JDBC connection string was created manually.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input/Output Settings

See JDBC Driver Settings for input and output settings.

Oracle Database | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/sql/oracle_database/

Oracle Database

Connection Settings

Database

The name of the database to connect to. By default, if no identifier style is set, the connection assumes an SID style.

If you are connecting using a service name, prefix it with a forward slash (`/`).

Examples of valid database field entries:

- **SID Connection (Default):** `SID`
- **SID Connection:** `:SID`
- **Service Name Connection:** `/ServiceName`

Username

Username for authentication with the database.

Password

Password for authentication with the database.

Additional JDBC Options

Additional JDBC options to include in the JDBC connection string. This is Oracle JDBC specific. Options are entered exactly as they would be if the JDBC connection string was created manually.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

Type

The type of input to use. The options are *Query* and *CDC*.

- When using a *Query* input, users can send a query to the corresponding database.
- When using a *CDC* input, users can track changes on a specified table

Query Settings

Sends the query to the database and returns the generated result, if there is any.

Query

The full SQL query, which can include any SQL syntax including stored procedures.

Index

This setting is used to index the SQL query and only get new data on each query. When enabled, specifies a Column Name and default Value to reference within the query. This name/value pair is cached across successive product runs and allows the next query to pick up where it left off.

Note the Column Name must be the column name in the table that is being used for indexing. An example query might be SELECT * FROM SimpleMachineX WHERE id > {{index}} By enabling indexing and specifying the Column Name as id and the Value as zero, the first time the query is run {{index}} is set to zero and after the query {{index}} is updated to the latest value from id. The latest value is cached to disk. Successive queries will use the latest value cached and then update the cached value on success.

To reset the index, navigate to the “Index” page and delete the index value. Note that test reads on the input done from the UI will use the current index Value but will not update the index Value.

Column Name

Specifies the column name used to index. To update the index, the column name must be returned in the SQL query.

Value

Specifies the initial value used for the index.

CDC Settings

Generates a standardized value that represents a SQL change event.

Oracle CDC changes are captured using the Oracle Log Miner. To ensure changes are tracked as expected, take the necessary steps to ensure:

- Archive Logging is enabled
- User has necessary permissions
- Supplemental logging is set to capture expected changes
- Tracked table definitions are not altered

```
{
    "table": {
        "schema": "public", // The schema name for the tracked table
        "name": "cdc_table" // The name of the tracked table
    },
    "operation": "Update", // Can be Insert, Update, or Delete
    "primaryKey": {
        // Contains each primary key that belongs to the row along with their corresponding
    },
    "before": {
        // The state of the row before the operation occurred
        // Is available for Update and Delete operations
    },
    "after": {
        // The state of the row after the operation occurred
        // Is available for Update and Insert operations
    },
    "position": 3662432 // The SCN of the retrieved CDC transacation
}
```

Insert Operation

```
{
    "table": {
        "schema": "public",
```

```
        "name": "cdc_table"
    },
    "operation": "Insert",
    "primaryKey": {
        "machineId": 1
    },
    "after": {
        "machineId": 1,
        "speed": 0.532
    },
    "position": 3662430
}
```


Update Operation


```
{
    "table": {
        "schema": "public",
        "name": "cdc_table"
    },
    "operation": "Update",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.532
    },
    "after": {
        "machineId": 1,
        "speed": 0.567
    },
    "position": 3662432
}
```


Delete Operation


```
{
    "table": {
        "schema": "public",
        "name": "cdc_table"
    },
    "operation": "Delete",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.567
    },
    "position": 3662436
}
```


Table Names

A list of table names to track changes for.

Transaction Count

The max number of transaction (SCN) changes that are retrieved per read. Each transaction (SCN) could have multiple changes/rows associated with it. Setting the maximum transactions to zero will read all available transactions.

Starting Transaction

Defines the timestamp that marks where change data capture (CDC) begins. The value can be provided as an ISO-8601 string or a Unix epoch timestamp in milliseconds. This value is automatically converted to a relative SCN for Oracle LogMiner. If no value is provided, the system starts from the earliest available SCN.

Output Settings

Table

The name of the SQL table or stored procedure to write to. The table must already exist and have column names that match the model attribute names. Table names containing capital letters must be placed inside quotes (e.g., "MyTable").

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are **Off**, **Create**, and **Create & Update**.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. All tables are created with an auto incrementing _id column used as the primary key. Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 2048 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR2(2048)
DateTime	TIMESTAMP
Boolean	CHAR

Write Data Type	SQL Data Type
UInt8	NUMBER(3)
Int8	NUMBER(3)
UInt16	NUMBER(5)
Int16	NUMBER(5)
UInt32	NUMBER(10)
Int32	NUMBER(10)
UInt64	NUMBER(20)
Int64	NUMBER(20)
Real32	BINARY_FLOAT
Real64	BINARY_DOUBLE

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when Create Table is set to Off. Disable this refresh by setting the time interval to 0.

PostgreSQL | User Guide

Source: https://guide.highbyte.com/configuration/connect/connections/sql/postgresql/

PostgreSQL

Connection Settings

Database

Name of the database to connect to.

Username

Username for authentication with the database.

Password

Password for authentication with the database.

Additional JDBC Options

Additional JDBC options to include in the JDBC connection string. This is PostgreSQL specific. Options are entered exactly as they would be if the JDBC connection string was created manually.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input Settings

Type

The type of input to use. The options are *Query* and *CDC*.

- When using a *Query* input, users can send a query to the corresponding database.
- When using a *CDC* input, users can track changes on a specified table

Query Settings

Sends the query to the database and returns the generated result, if there is any.

Query

The full SQL query, which can include any SQL syntax including stored procedures.

Index

This setting is used to index the SQL query and only get new data on each query. When enabled, specifies a Column Name and default Value to reference within the query. This name/value pair is cached across successive product runs and allows the next query to pick up where it left off.

Note the Column Name must be the column name in the table that is being used for indexing. An example query might be SELECT * FROM SimpleMachineX WHERE id > {{index}} By enabling indexing and specifying the Column Name as id and the Value as zero, the first time the query is run {{index}} is set to zero and after the query {{index}} is updated to the latest value

from id. The latest value is cached to disk. Successive queries will use the latest value cached and then update the cached value on success.

To reset the index, navigate to the "Index" page and delete the index value. Note that test reads on the input done from the UI will use the current index Value but will not update the index Value.

Column Name

Specifies the column name used to index. To update the index, the column name must be returned in the SQL query.

Value

Specifies the initial value used for the index.

CDC Settings

Generates a standardized value that represents a SQL change event.

```
{
    "table": {
        "schema": "public", // The schema name for the tracked table
        "name": "cdc_table" // The name of the tracked table
    },
    "operation": "Update", // Can be Insert, Update, or Delete
    "primaryKey": {
        // Contains each primary key that belongs to the row along with their corresponding
    },
    "before": {
        // The state of the row before the operation occurred
        // Is available for Update and Delete operations
    },
    "after": {
        // The state of the row after the operation occurred
        // Is available for Update and Insert operations
    }
}
```

Insert Operation

```
{
    "table": {
        "schema": "public",
        "name": "cdc_table"
    },
    "operation": "Insert",
    "primaryKey": {
        "machineId": 1
    },
    "after": {
        "machineId": 1,
        "speed": 0.532
    }
}
```

Update Operation

```
```

```
{
    "table": {
        "schema": "public",
        "name": "cdc_table"
    },
    "operation": "Update",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.532
    },
    "after": {
        "machineId": 1,
        "speed": 0.567
    }
}
```

Delete Operation

```
{
    "table": {
        "schema": "public",
        "name": "cdc_table"
    },
    "operation": "Delete",
    "primaryKey": {
        "machineId": 1
    },
    "before": {
        "machineId": 1,
        "speed": 0.567
    }
}
```

Slot Name

The name of the replication slot where change events are written to from the WAL (Write Ahead Log).

Publication Name

The name of the publication used to filter change events from the replication slot.

Transaction Count

The max number of transaction (LSN) changes that are retrieved per read. Each transaction (LSN) could have multiple changes/rows associated with it.

Output Settings

Table

The name of the SQL table or stored procedure to write to. The table must already exist and have column names that match the model attribute names. Table names containing capital letters must be placed inside quotes (e.g., “MyTable”).

Write Type

Writes to the table can be Inserts, Updates, or Upserts. Inserts add the output data as new rows to the table. Updates update existing rows that have a matching value of the attribute referenced in the Where Column. If there are no matches Update does nothing. Upsert performs an Update if there are matching rows, and if there are no matching rows it performs an Insert.

Where Clause

Specify the body of the where clause used to filter updated records. This setting is used if the Write Type is set to Update or Upsert. As an example, assume the table being to contains a column 'batchId'. To only update rows that contain a 'batchId' greater than 15, the Where Clause would be set to "batchId > 15". To specify multiple columns and conditions users can use conditional logic such as "AND" and "OR". This field supports dynamic outputs.

Log as JSON

Specifies whether modeled values should be serialized as a single column JSON blob versus expanding to multiple columns. When enabled, the column name is 'json'.

Create Table

Controls how the table definition is managed. Options are **Off**, **Create**, and **Create & Update**.

- When **Off** is used, it's assumed that the table schema is defined and managed externally. The table must exist. See Table Cache Interval for details on how to respond to table schema changes.
- When **Create** is used, it's assumed the connection manages the table and it's created once. If the table doesn't exist it's created using the schema of the payload.
- When **Create & Update** is used, it's assumed the connection manages the table and updates the table on schema change. For example, if a new attribute is added to the data, the column is added to the table.

When creating a table the data from the write is used to control the table schema. All tables are created with an auto incrementing _id column used as the primary key. Tables will also include _name, _model, and _timestamp columns unless the output filters these out. Below is a mapping of write data types to SQL data types. Note strings are limited in size to 2048 characters. This must be changed by altering the table after creating if larger strings are required.

Write Data Type	SQL Data Type
String	VARCHAR(2048)
DateTime	DATETIME
UInt8	SMALLINT
Int8	SMALLINT
UInt16	INTEGER
Int16	SMALLINT
UInt32	BIGINT
Int32	INTEGER
UInt64	NUMERIC
Int64	BIGINT
Real32	REAL
Real64	FLOAT

Table Cache Interval

This setting is useful for scenarios where there have been external modification to the table schema, and those updates should be reflected in the internally cached table schema. This setting is only available when Create Table is set to Off. Disable this refresh by setting the time interval to 0.

SQLite | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/sql/sqlite/

SQLite

The SQLite Connection allows for creating and interacting with local DB files or in memory storage. The connection can be used to manage local state in the Intelligence Hub.

Below are details on the connection, input, and output settings and supported features.

Connection Settings

Database

The name of the database file. For example `mydatabase.db` creates a file in the runtime directory named `mydatabase.db`. Database can include a path (ex. `C:\mydatabase.db`). In memory databases, which are not persisted between runs, are also supported by using `:memory:`.

Keep Alive

See JDBC Driver Settings for keep alive settings.

Input/Output Settings

See JDBC Driver Settings for input and output settings.

Webhook | User Guide

> Source: https://guide.highbyte.com/configuration/connect/connections/webhook/

Webhook

The Webhook connection listens on a port for incoming HTTP messages and transforms and filters these messages making the data available as an input in the server. For example, a webhook input might received JSON messages from a work order system, and filter only work orders that are completed.

Connection Settings

Port

The port that the webhook listens on. This port must be open in the firewall for other applications to connect and send HTTP requests to.

User HTTPS

When enabled the webhook is available on `https://<hostname>:port`

Schema Certificate

The private key used to encrypt the webhook when HTTPS is enabled.

Authorization Type

Controls if authentication is required for the connection. When set to None, not authentication is required. When set the Bearer, the HTTP client must include the Authorization Token in the header of the request.

Authorization Token

A custom token that clients must include in the HTTP header to authenticate in the following format.

- `Authorization: Bearer <token>`

Input Settings

Inputs provide a way to filter messages sent to the webhook. The webhook accepts the following content types:

- `application/json`
- `application/xml`
- `application/octet-stream`
- `text/plain`
- `application/x-www-form-urlencoded`

Filter

The filter option filters incoming payloads using data in the payload. Click 'Add Value' to add an attribute Name and Value to the filter. Multiple Name/Value pairs are logically AND together. Matching is case sensitive.

As an example, this filter will match the example payload below.

Name	Value
Machine.Online	true
Machine.AssetInfo.ID	MACH-L1-102

```
// Example JSON payload
{
  "Machine" : {
    "AssetInfo" : {
      "ID" : "MACH-L1-102",
      "SN" : "B320-3R821N9-D",
      "Date" : "2018-09-29"
    },
    "Online" : true,
    "Utilization" : 40.0
  }
}
```

Include Metadata

Enable this to include the headers in the payload. When enabled header name:value pairs are placed at the root of the payload and the body is included under a value attribute.

```
{
  "header1": "header1value",
  "header2": "header2value",
  "value": {
    "tag1": 100
  }
}
```

Output Settings

Outputs are not supported

Inputs | User Guide

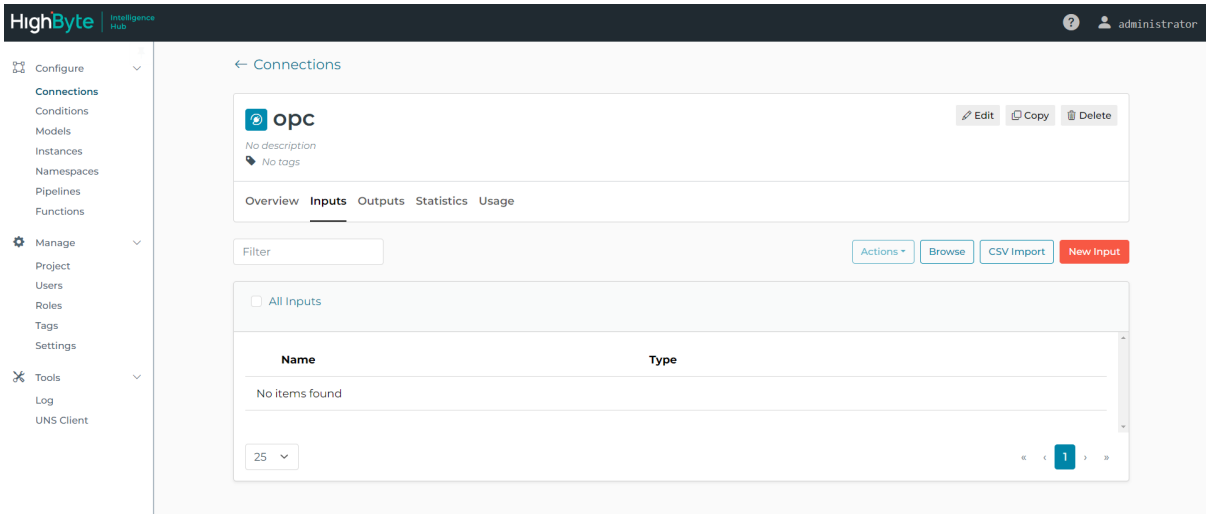
> Source: https://guide.highbyte.com/configuration/connect/inputs/

Inputs

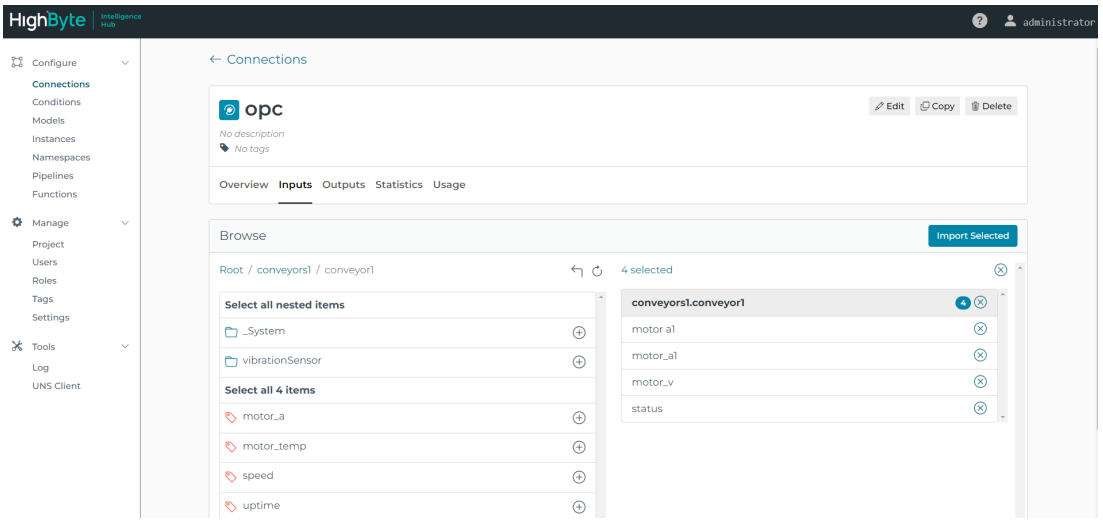
An Input represents a readable path to a data point on a Connection. This section covers general settings and configuration across all Inputs. See Connections for protocol specific settings.

Create an Input

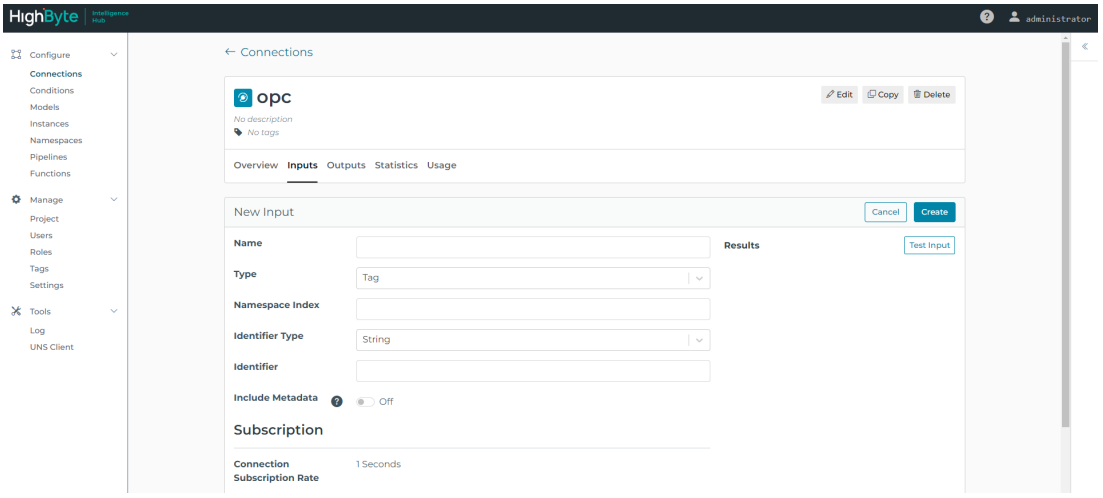
1. Navigate to *Connections* in the configuration's Main Menu and select a connection. Inside the selected connection navigate to the *Inputs* menu item. Here you can add, edit, and view inputs.



2. Depending on the type of connection you may be able to *Browse* and import one or more Inputs (e.g., OPC UA). Alternatively, you can click the *New Input* to manually configure an input.
    - Click the *Browse* button to *View* and *Select* the desired inputs. When finished, click the *Import Selected* button.



    - Click the *New Input* button to manually configure an input.



3. Set any *General* and *Protocol Specific Settings*. Click *Submit* to add the input and return to the input list. General Settings are described below.

All inputs have the following general settings

General Settings

Name

Specifies the name of the input. The name must be unique across all inputs for the specified connection. Names can only contain alphanumeric and underscore characters (e.g., A-Z, a-z, 0-9 or _).

Cache

When enabled, specifies how long the input's value is cached and reused internally after a successful read from the underlying connection. Once the cached value's lifetime expires, the cache is invalidated and a new read is issued to the underlying connection. The lifetime interval is specified in milliseconds, seconds, minutes, hours, or days. Valid range is between 10 milliseconds and 36 days. *Note: The cached value and its lifetime is persisted to disk and used across product runs until the value expires.*

Parameters

Input parameters provide a way to make generalized inputs that can return different data when called. For example a SQL input can take a workOrderID parameter to return the information for a specific work order.

Please refer to the Parameters for details.

Templating

Input templates help reduce configuration when many inputs have similar settings with only minor differences.

An example of this is an OPC UA server that has tag groupings that represent motors where each is represented by multiple tags whose address only differs by a motorID. In this case, a single OPC UA Input can be created with a parameter (motorID) with an appropriate value range to create multiple inputs.

Please refer to the Templating for details.

Input Indexing

Some inputs support indexing to read historical data. For example, indexing can be used with SQL reads to only read new records from a table. Indexing is setup in the Connection specific input settings, but the current value of an Index can be viewed or cleared under the Index menu for the input.

Test Reads

All inputs support test reads using the “Read Input” button. Test reads perform a read using the current input settings, and return a result in JSON format. Test reads do not impact index caching, time out after 60 seconds. Reads greater than 256KB are truncated and a partial result is returned.

Dynamic References

This is a legacy feature that is still supported, but it’s recommend to use Parameters instead.

Inputs can reference other Inputs or Instances in their configuration. An example of this is a SQL input that references a machineID from an OPC UA input to pull maintenance data for the machine. The syntax for referencing another input is as follows. When this input is read, the referenced input is read first and the result is inserted into the query. If the referenced input read fails, the input read will also fail. Dynamic references are not supported on all input fields. Support is specific to each connection type.

```
SELECT * FROM machineTable WHERE machineID={{Connection.opc.machineIDTag}}
```

Protocol Specific Settings

See Connections for Protocol Specific Settings.

Outputs | User Guide

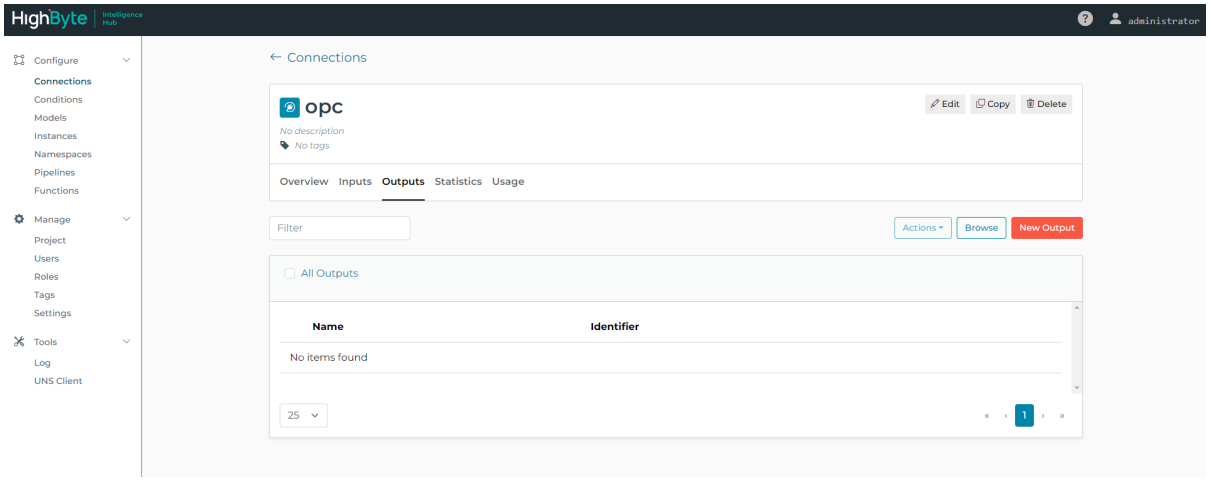
> Source: https://guide.highbyte.com/configuration/connect/outputs/

Outputs

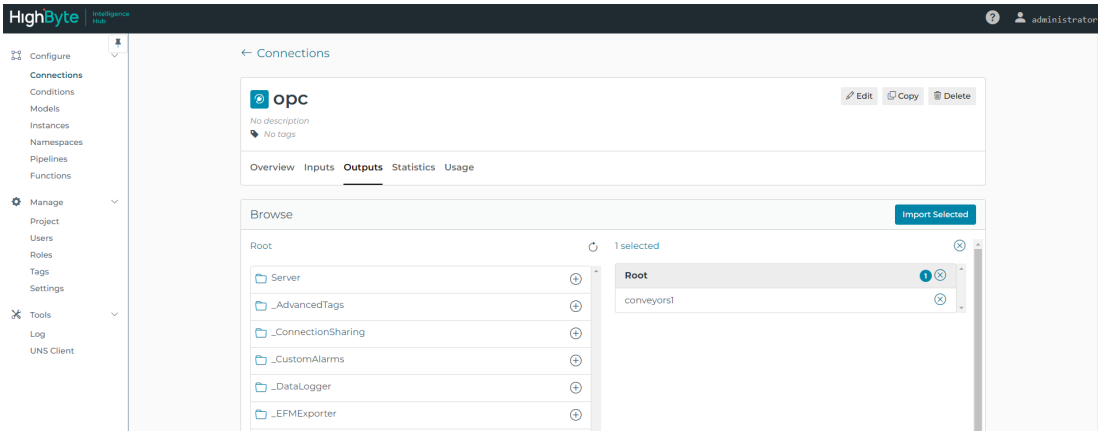
An Output represents a writeable path to a data point on a connection. Examples of outputs include OPC UA tags, MQTT topics, SQL tables, etc. This section covers general settings and configuration across all Outputs. See Connections for protocol specific settings.

Create an Output

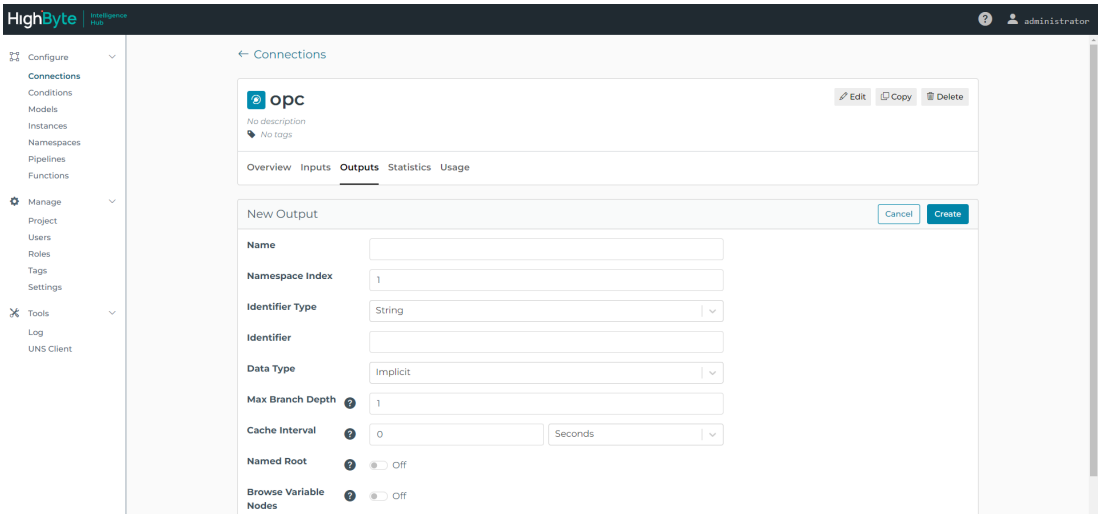
1. Navigate to *Connections* in the configuration's Main Menu and select a connection. Inside the selected connection navigate to the *Outputs* menu item. Here you can add, edit, and view outputs.



1. Depending on the type of connection you may be able to *Browse* and import one or more Outputs (e.g., OPC UA). Alternatively, you can click the *New Output* to manually configure an output.
    - Click the *Browse* button to *View* and *Select* the desired outputs. When finished, click the *Import Selected* button.



    - Click the *New Output* button to manually configure an output.



2. Set any *General* and *Protocol Specific Settings*. Click *Submit* to add the output and return to the output list. General Settings are described below.

General Settings

Name

Specifies the name of the output. The name must be unique across all outputs for the specified connection. Names can only contain alphanumeric and underscore characters (e.g., A-Z, a-z, 0-9 or _).

Breakup Arrays

When enabled, arrays sent to this output are broken up into individual objects and sent one at a time. One example of this is breaking up SQL rows over multiple MQTT topics.

Note: Some connectors, like SQL, can efficiently process arrays via bulk operations. In this case enabling breakup arrays decreases performance.

Attribute Filter

Specifies attributes to remove from a complex payload. Nested attributes are excluded by defining the attribute path (delimited by the dot (.) character).

For example, to remove the field “temp” from the payload below, use the attribute path “value.temp”.

```
{ "name": "myInstance", "value": { "temp": 123, "pressure": 456 } }
```

The result of the filtering the attribute path “value.temp” is:

```
{ "name": "myInstance", "value": { "pressure": 456 } }
```

Note: Only complex payloads are filtered and invalid attribute paths are ignored. When outputting an array of complex objects, attribute filters are applied to each object in the array.

Dynamic Output Settings

Often times you may want to change output settings based on the data being sent. An example of this would be setting the MQTT topic depending on data in the write payload. There are two ways to do this:

1. Using a Write New Stage stage in a Pipeline (recommended)
2. Using a Write Stage, and defining the Output settings using the {{this.payloadAttribute}} syntax

Below is an example using the Write Stage. Assume an MQTT topic on the output is set to the following:

```
{{this.siteID}}/{{this.areaID}}/{{this.machineID}}
```

If the data written to a MQTT output looks as follows:

```
{
    "siteID": "Portland",
    "areaID": "Engineering",
    "machineID": "Boiler1"
}
```

The topic would be set to `Portland/Engineering/Boiler1` for the write.

Dynamic outputs are supported on any text output fields. If a reference made is not in the payload (e.g., `{{this.badAttribute}}`) the operation fails with an appropriate error message.

Test Writes

All outputs support test writes. Test writes output an expression result to the output. The `Expression to write` can be JSON object, single value, or a source reference (e.g., Input, Instance, Condition, etc.).

Some examples of test write expressions include:

```
// Outputs a single value
1
```

```
// Outputs a string
"hello world"
```

```
// Outputs an instance
{{Instance.myInstance}}
```

```
// Outputs a JSON object
var out = {
    "myObject": {
        "tag1": 123,
        "tag2": "hello world"
    }
}
out
```

Protocol Specific Settings

See Connections for Protocol Specific Settings.

Model | User Guide

Source: https://guide.highbyte.com/configuration/model/

Model

Modeling is fundamental to HighByte Intelligence Hub. Modeling allows for pulling data from many discrete data sources (e.g., OPC UA, SQL, REST) and combining that data into useful information in a common format. Upstream data consumers can then rely on standardized information regardless of the underlying data sources, which can differ greatly per location. The Intelligence Hub breaks modeling into Models and Instances.

Models represent the definition of information. For example, a Motor model may include data associated with volts, amps, and on/off state.

Model Import can be used to create models from JSON, existing inputs, or other standard definitions.

Instances represent an instantiation of a model and pulls together the various data sources that represents something real (e.g., System, Process, Asset, Role, etc.). In the case of the Motor model above, a factory with 100 motors would model each as a unique Instance.

Instances | User Guide

Source: https://guide.highbyte.com/configuration/model/instances/

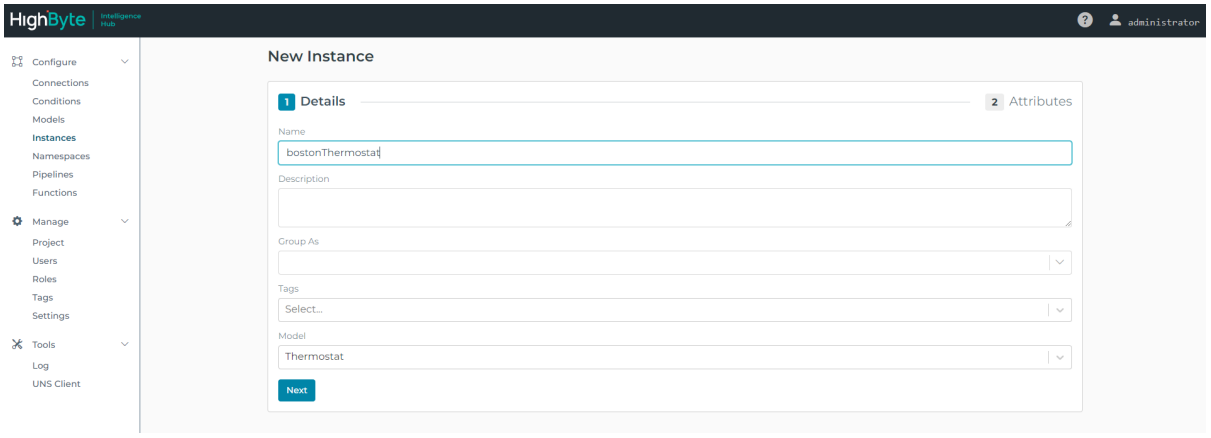
Instances

Whereas a *Model* is the standard definition of something (asset, process, etc), an *Instance* is the actual thing. For example, a *Model* for a line may define a quality managers view of the line (scrap, good count, bad count). If there are 10 lines, there would be 10 instances, each populated with the data sources specific to the line.

Create an Instance

1. Navigate to *Instances* in the configuration's Main Menu. Here you can add, edit, and view instances. To add a instance, click the **New Instance** button.

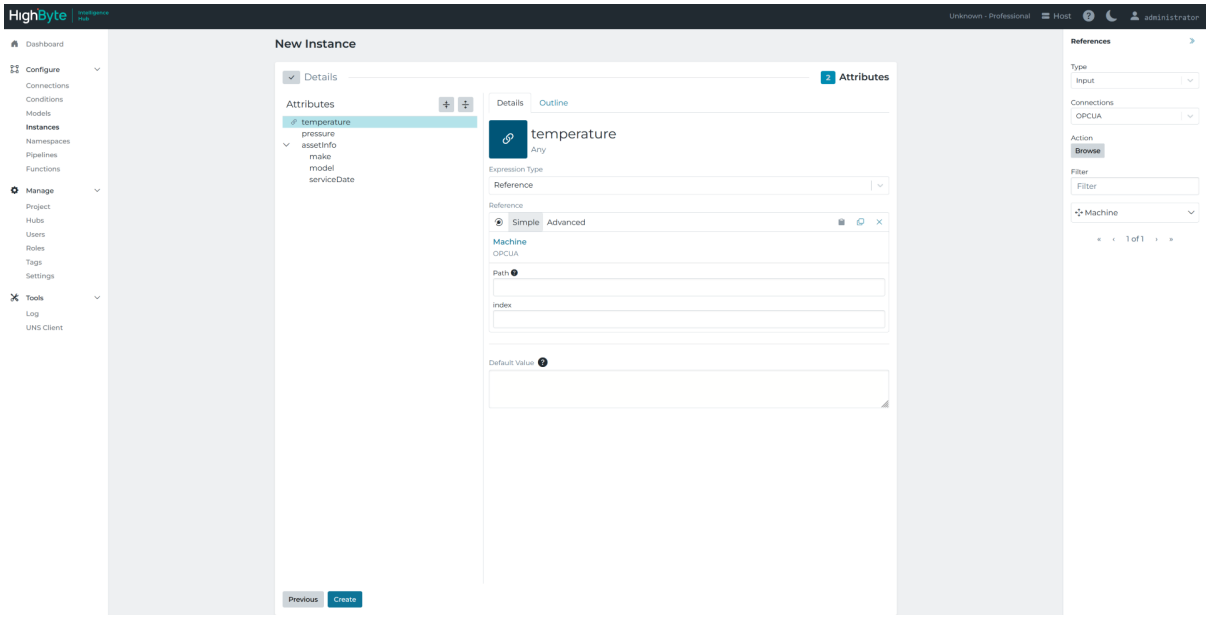
2. Enter a *Name* of the instance and the *Model* it's based on. This will automatically pre-populate the attributes to be assigned to this instance. Optionally enter in a *Description* and a *Group As* folder to categorize the instance. Click *Next* to continue.



3. The Model attributes are shown in the tree. Use the tree and Detail and Outline tabs to configure each attribute. Use the *References* panel to find and select sources of interest and drag them directly on the expression or reference field, or into the tree. More information on attribute configuration is provided below. Click the *Submit* button when finished.

Attribute Configuration

The tree view shows the attributes for the parent model and all child models. Each attribute has the following settings.



Expression Type

This controls how the attribute is processed. The two options are Reference and Expression.

Reference

For simple cases, like mapping an attribute to an OPC UA tag value, use a Reference type. Reference types don't require JavaScript to execute when reading the instance and are faster.

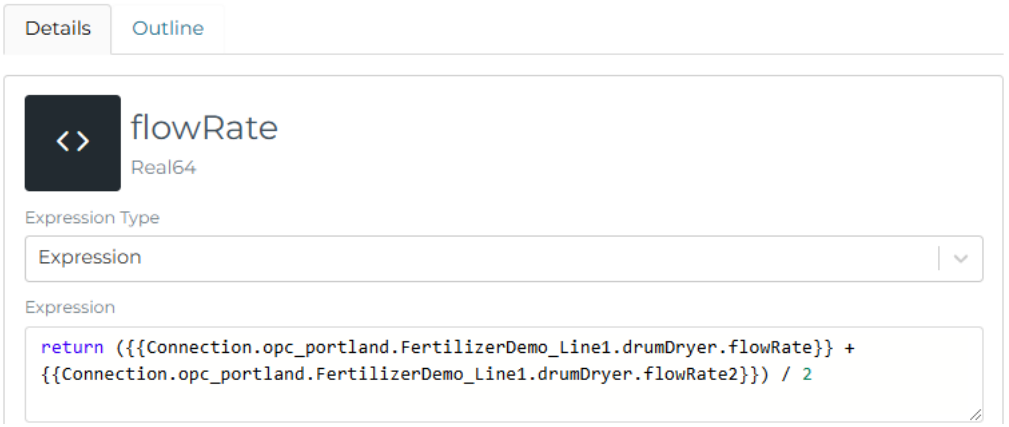
See References for more details.

Expression

Use the Expression type for attributes that require JavaScript. This could be for scaling a value, combining multiple values, or applying other logic. These attributes require the use of the JavaScript engine which has slight overhead. These attributes also require an explicit return statement.

Drag & drop references from the reference panel, and write JavaScript as needed. Hover over and click the expansion icon in the top right of the expression box or enter CTL-SHIFT-F while inside an expression to expand the expression view and provide more area for writing expression.

See Expressions for more details.



Reusing JavaScript Snippets

In some cases, there are snippets of JavaScript code that get reused in many expressions across many Instances.

In these situations Functions can be used to reduce duplicate code.

Default Value

The default value is used if the expression or reference field are empty or fail to compute a result. If there is no default value the default model attribute value is used.

Default values are represented as JSON. Simple number values can be entered as 123. String must be quoted. Complex objects like {"attribute": 123} are also supported.

Attribute Hierarchy

Object and Modeled attribute types are shown as hierarchy in the tree. The root node can be set to Reference or Expression type. When reading the Instance, the root node is evaluated first. If it returns hierarchy the hierarchy is expanded into the child attributes by name using case sensitive matching. Child attributes can then override the value from the parent attribute. Below is an example.

Consider the following hierarchy in an instance, coming from an attribute of type Object or Modeled.

```
{
  "lineMotor" : {       // Mapped {{Instance.line1Motor}}
    "speed": "",        // left blank
    "location": "line1" // default value set to line1
  }
}
```

Assume {{Instance.line1Motor}} produces the following data when read.

```
{
  "speed": 100,
  "location": "test",
  "someOtherAttribute": "ignore"
}
```

When the parent instance is read, {{Instance.line1Motor}}.speed maps to the child attribute. {{Instance.line1Motor}}.location is overridden. The other attribute is ignored as it has no mapping. The result is as follows.

```
  "lineMotor" : {
    "speed": 100,
    "location": "line1"
  }
```

Parameters

Instance parameters provide a way to make generalized instances that can return different data when called. For example an Instance can represent a type of machine and a machineID parameter can be passed as a parameter to control what machine to read.

For more details on how to use parameters with instances, please see the Parameters section.

Advanced

Initialization

This setting is only available when Legacy Execution Mode is disabled. The Initialization block is the first thing that executes when the instance is read. Use this to read common data sources that are required in the instance, or to initialize default values that are used in multiple attributes.

Example of an initialization block to read a source and set defaults values.

```
let mesData = {{Connection.mes.machineStatus({"machineid":"10"})}}
variables.machineName = mesData.name
variables.defaultSpeed = 100
```

These can then be referenced in Expression attributes as follows.

```
return (variables.machineName)
```

Templating

Instances can be templated to quickly create many Instances that have the same basic configuration.

See Templating for more details.

Expand Arrays

The Expand Arrays feature is useful in cases where the source data is an array of objects (e.g. SQL rows) and you want to transform each row through the model and output a transformed array. Below is an example of how to accomplish this.

Consider the source input {{Connection.sql.rows}}, which returns the following data.

```
[
  {
    "timestamp": 123,
    "value": 100
  },
  {
    "timestamp": 456,
    "value": 101
  }
]
```

Configure the Instance as follows, where the time and rpm attributes are mapped to index [0] of the array. Index [0] is important, because the Expand Array feature breaks the input array into N arrays of size 1 (i.e. index [0]) and runs each array through the Instance.

```
{
  // Note this format is provided as an example, and this configuration must be done through
  "time": {{Connection.sql.row}}[0].timestamp,
  "rpm": {{Connection.sql.row}}[0].value,
  "model": "myModel"
}
```

The resulting Instance read looks as follows, with each row run through the Instance.

```
[
    {
      "time": 123,
      "rpm": 100,
      "model": "myModel"
    },
```

```
    {
      "time": 456,
      "rpm": 101,
      "model": "myModel"
    }
]
```

Note it's possible to combine data from multiple arrays using this feature, but each source array must have the same number of rows.

Legacy Execution Mode

Legacy Execution Mode has been deprecated as of version 4.1 and will be removed in a future release. It is recommended that all Instances migrate away from using this feature.

Controls how the instance is read. When enabled, Instance are read like they were in version 3.x and older. For new instance, leave this setting disabled.

In version 3.X and older, instance attributes were calculated using a unique JavaScript environment for each attribute. This approach isolated the execution of each attribute, but could significantly impact performance when an instance had a lot of attributes. Expressions also did not require return statements, making it confusing to understand what an expression returned when the expression included complex JavaScript code.

When this setting is disabled, the instance uses a single JavaScript context to execute all expressions, resulting in significant speed gains. Expression also must have return statements. For simple attributes use the Reference type, but any attribute set to an Expression type must include a return in the expression. When disabled, the Initialization block is also available.

Migrating Legacy Instances

To migrate a legacy instance a return statement must be added to the expression.

Additionally, model schema validation has been improved for new instances. It is recommended that a migrated instance is retested to ensure that the values produced do not cause an error with the new (more strict) schema validation.

The model validation for new instances is consistent with the Modeling and Model Validation stages in pipelines.

Instance Metadata in Expressions

Metadata about the Instance is available for use inside the Instance expressions. Below are examples of the available metadata and how to reference it.

Instance Metadata	Description
`instance.name()`	Returns the name of the instance. For templated instances this returns the templated name.
`instance.model()`	Returns the model name of the instance.
`instance.usage()`	Returns an array of usage paths for the instance. For example, if instance machine1 has a parent instance line, which is part of a plant/area hierarchy, than `instance.usage()[0]` returns the full path to the instance `plant/area/line/machine1`. The usage method takes an optional delimiter (ex. `instance.usage("/")`).

Expressions | User Guide

Source: https://guide.highbyte.com/configuration/model/instances/expressions/

Expressions

An expression is simply the right-hand side of an assignment to the attribute it is attached to. It can be as simple as an assignment of a source, or as complex as a computation on one or more sources. The expression forms the basis for conditioning data.

The syntax for an expression is basic JavaScript. The Intelligence Hub uses GraalVM to execute the JavaScript in expressions.

Sample Expressions

Expression	Description
`{{Connection.OPC_Server.Room2BTSTAT_CV}}`	A simple assignment of the raw value associated with input *Room2B_TSTAT_CV* on connection *OPC_Server*.
`{{SQL.QueryAssetInfo.AssetID}}`	A complex data element assignment. Some connectors (e.g., REST, Sparkplug, SQL) return complex data values that contain one or more element values. In this example, a SQL connection named *SQL* has an input called *QueryAssetInfo* that returns a result set for which the element *AssetID* value is assigned.
`{{Connection.OPC_Server.LineAPartsProduced}} + {{Connection.OPC_Server.LineBPartsProduced}}`	Assigns the summation of two inputs named LineAPartsProduced and LineBPartsProduced on connection OPC_Server.
`{{Connection.OPC_Server.ProductID}}.split("-")`	Assuming the raw value of ProductID is a string value that is always formatted as follows: AA-BBBBB-C-DDD. Assigns the substring BBBBB from input *ProductID* on connection *OPC_Server*.
`Math.sin({{Connection.OPC_Server.DeviceSignal}})`	Calculates the sine of the raw value associated with input *Signal* on connection *OPC_Server* and assigns the result.
`{{Instance.ServerRoom_Thermostat}}`	A simple assignment of a modeled instance within a parent modeled instance definition.
`{{Instance.ServerRoom_Thermostat}}.Current > {{ServerRoom_Thermostat}}.Max`	A logical expression that tests if the value associated with attribute *Current* of instance *ServerRoom_Thermostat* is greater than the value associated with attribute Max of the same named instance.
`var data = [1,2,{{Connection.opc.tag}}]; data`	Build an array from a primitive tag
`{{Connection.opc.arraytag}}.slice(0,2)`	Return a part of an array.

```
// Build a complex value (JSON) from primitives
var value = {
  "employees": {
    "name": "steve",
    "value": {{Connection.opc.tag}}
    }
};
return (value)
```

```
// Iterate an array of complex data (CSV in this example) and return the row that meets a gi
var retRow = ({{Connection.csv.data}}).forEach(function (item, index) {
  if(parseInt(item["07A7"]) === 916){
    retRow = item;
  }
};
return (retRow)
```

Metadata in Expressions

For complex types passed to an expression (ex. Instances, SQL input, OPC UA Branch reads, etc) the following additional metadata is available in the expression.

Metadata	Description
`#model`	In the case of an Instance, this is the Model name. For inputs and other sources, it's empty.
`#name`	In the case of an Instance, this is the Instance name. For inputs and other sources, it's the source name.

Below is an example of how to get and set these values. These values appear by default in the output payload as `_model` and `_name` attributes.

```
// Get the values
var instanceName = {{Instance.myInstance}}["#name"]
var modelName = {{Instance.myInstance}}["#model"]

// Set the values. Changes the values of the default _name and _model output attributes
{{Instance.myInstance}}["#name"] = "changeName"
{{Instance.myInstance}}["#model"] = "changeModel"

// Change the values in a child object
{{Instance.myInstance}}.childInstance["#name"] = "changeName"
{{Instance.myInstance}}.childInstance["#model"] = "changeModel"
```

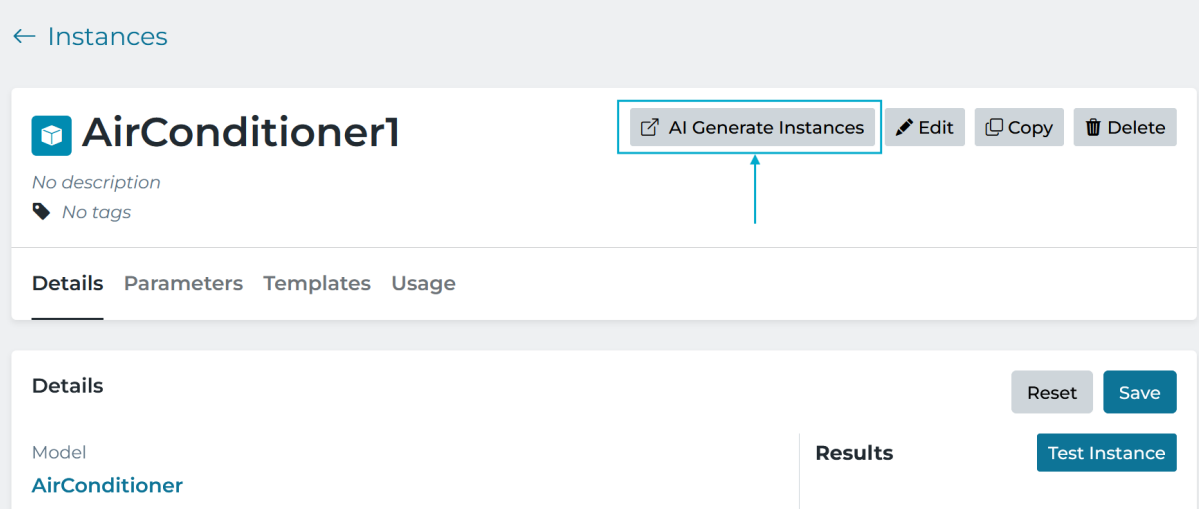
AI Generate Instances | User Guide

Source: https://guide.highbyte.com/configuration/model/instances/generate/

AI Generate Instances

AI Generate Instances is a tool designed to quickly create multiple instances using a large language model (LLM). This feature is ideal for when you need to generate many similar instances, such as when mapping a complex source namespace.

This feature requires an AI Connection with an input

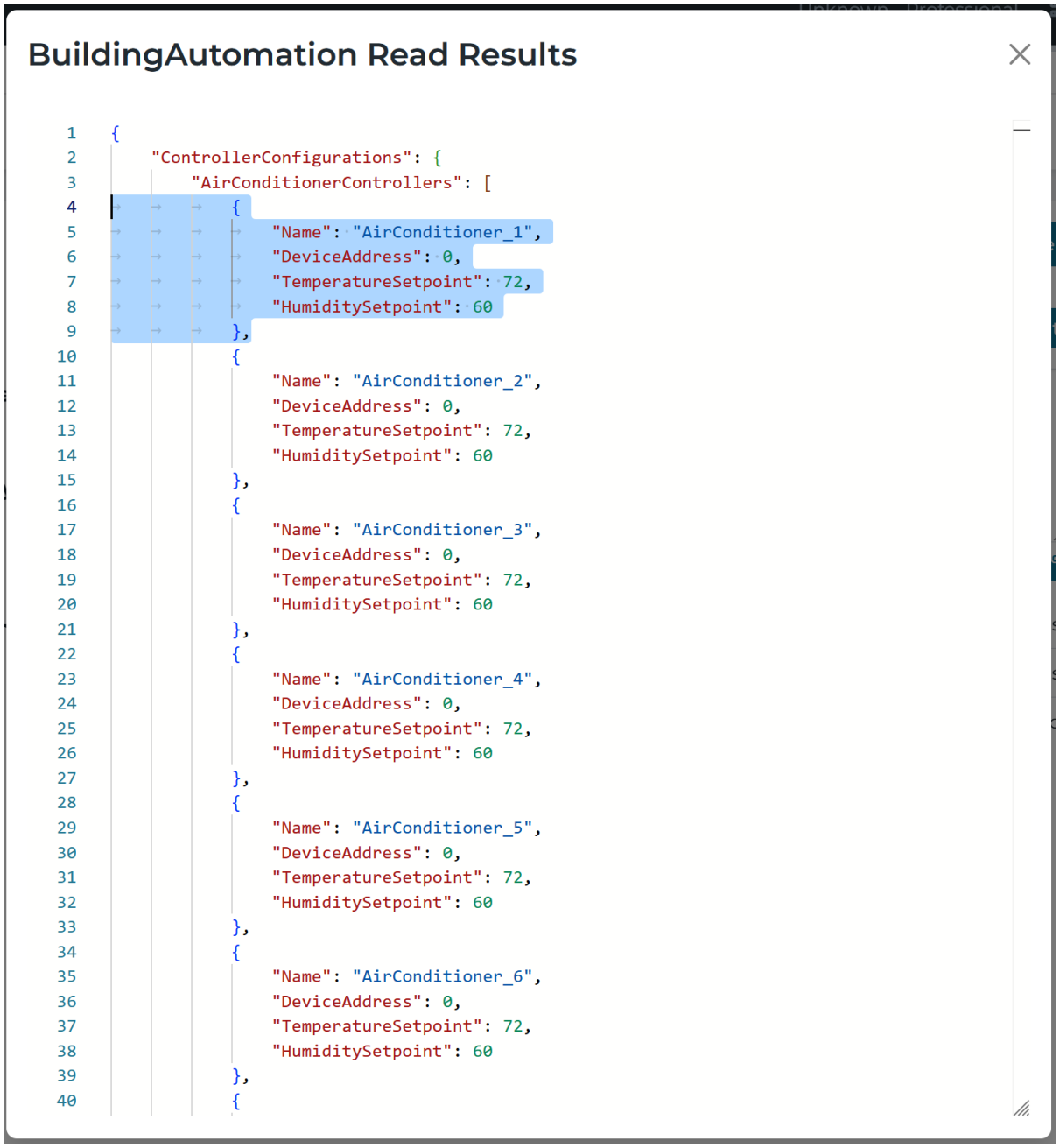


Using AI Generate Instances

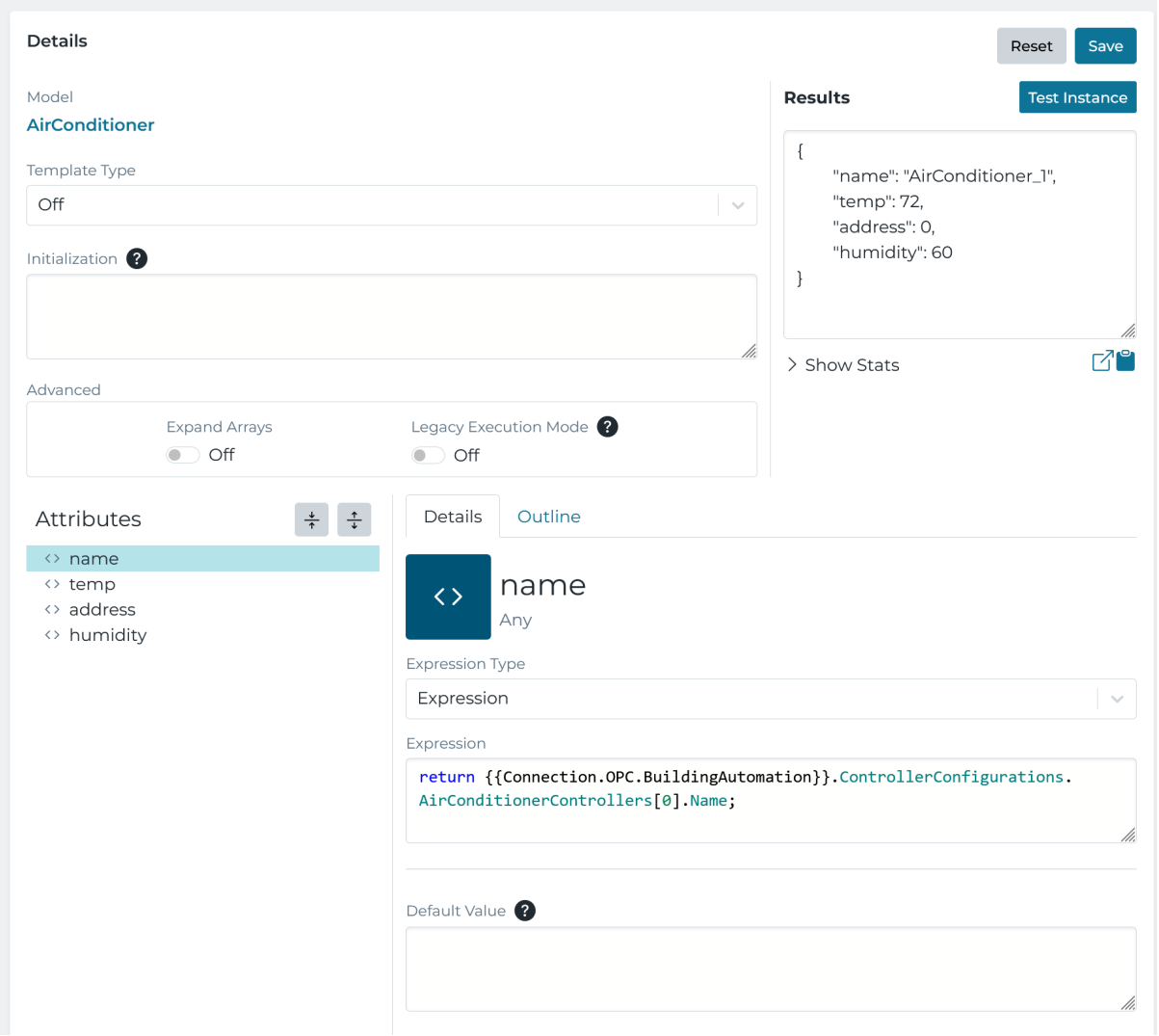
Create a Reference Instance

To generate instances you must first create an instance for the LLM to use as a reference.

This example will use the following data source that contains multiple air conditioner units. All of these units have a similar shape. Begin by mapping the highlighted section in to an instance.



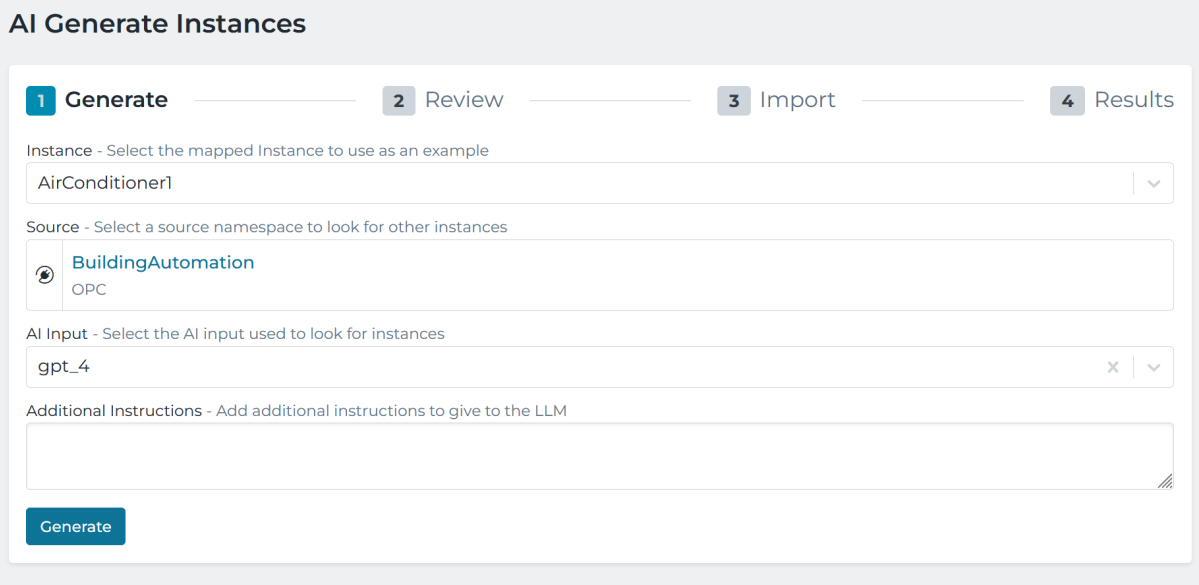
Once we've created an instance that maps to the first air conditioner, we can click the "AI Generate Instances" button to begin generating instances.



Generate

In the generate step we will need to provide a few things:

1. The instance that maps to a portion of the source.
2. The source that is being mapped.
3. An AI Input to use for the generation.
    - If this dropdown is empty an AI Connection must be setup AND that connection must have an input.
    - The instructions on the input will be ignored for the purpose of this tool.
4. (optional) Additional Instructions can be provided to guide the LLM.

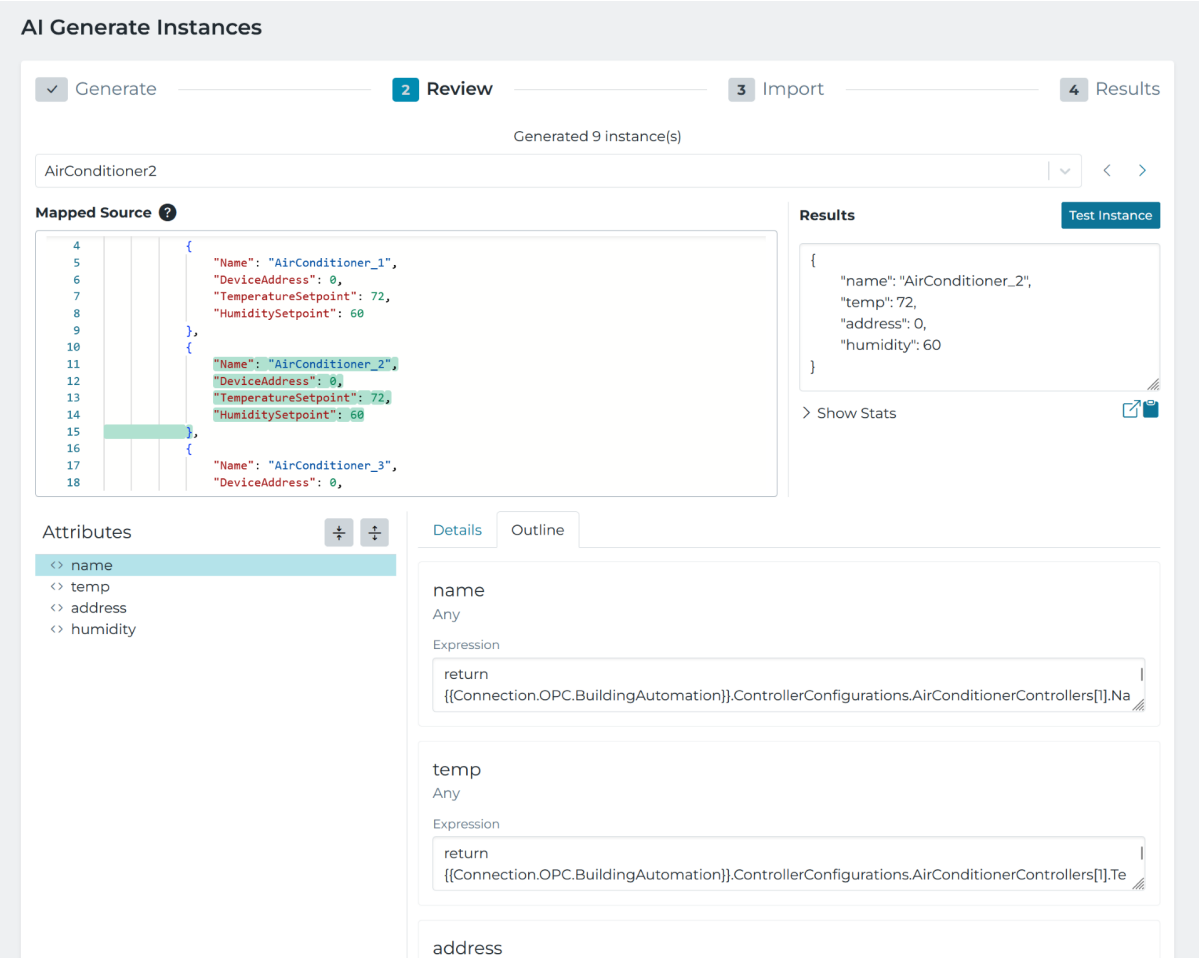


Review

Once the instances have been generated, the Review step provides an opportunity to review the instances before importing them.

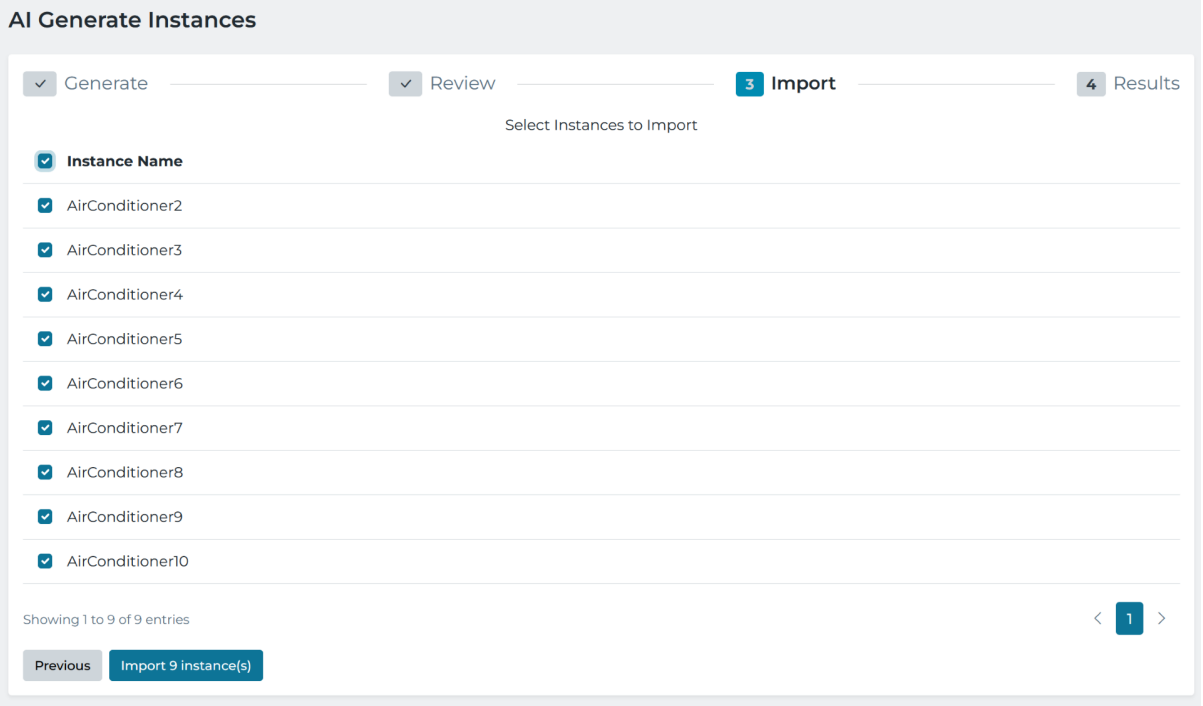
There will be a total number of instances generated at the top of the page, then a paginator below to inspect each of the instances.

Mapped Source - A projection of which part of the source namespace this instance maps to.
Test Instance - This can be used to perform a test read using the generated instance. This is useful to ensure the AI correctly generated the Intelligence Hub's reference syntax.
Attributes/Details - This section contains all of the details for the generated instance.



Import

After the instances have been reviewed, the Import step can be used to import the instances that were generated.

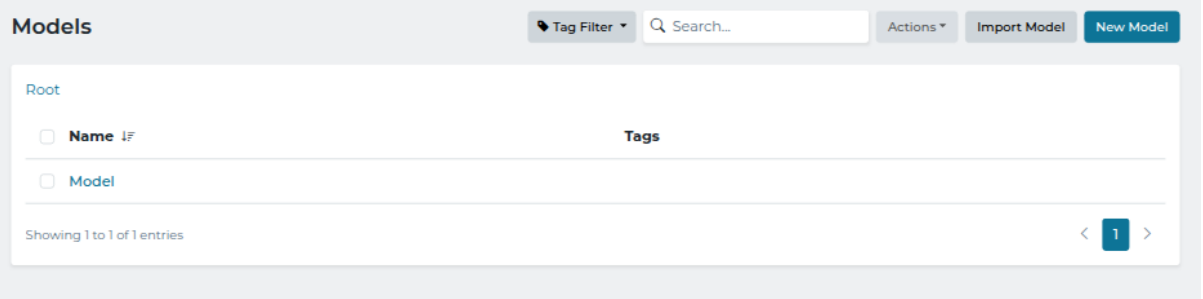


Model Import | User Guide

Source: https://guide.highbyte.com/configuration/model/model_import/

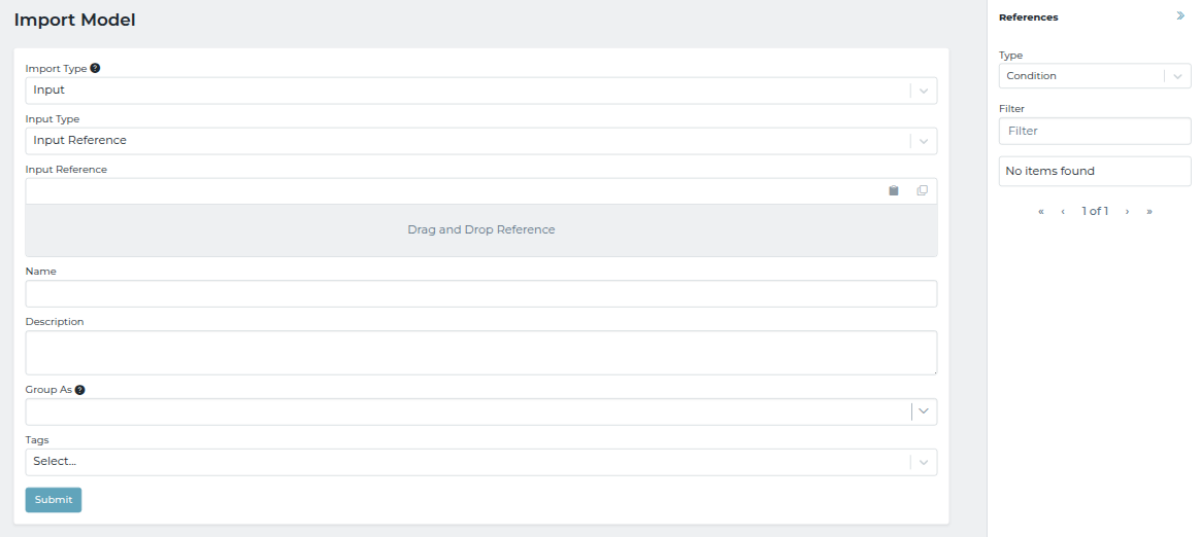
Model Import

Model Import can be used to create a model from a custom JSON payload, an existing Input, or an OPC UA Node Set. It can be performed by pressing **Import Model** from the Model list page. Existing Models are replaced when an existing Model with the same name as the imported Model(s) already exists.



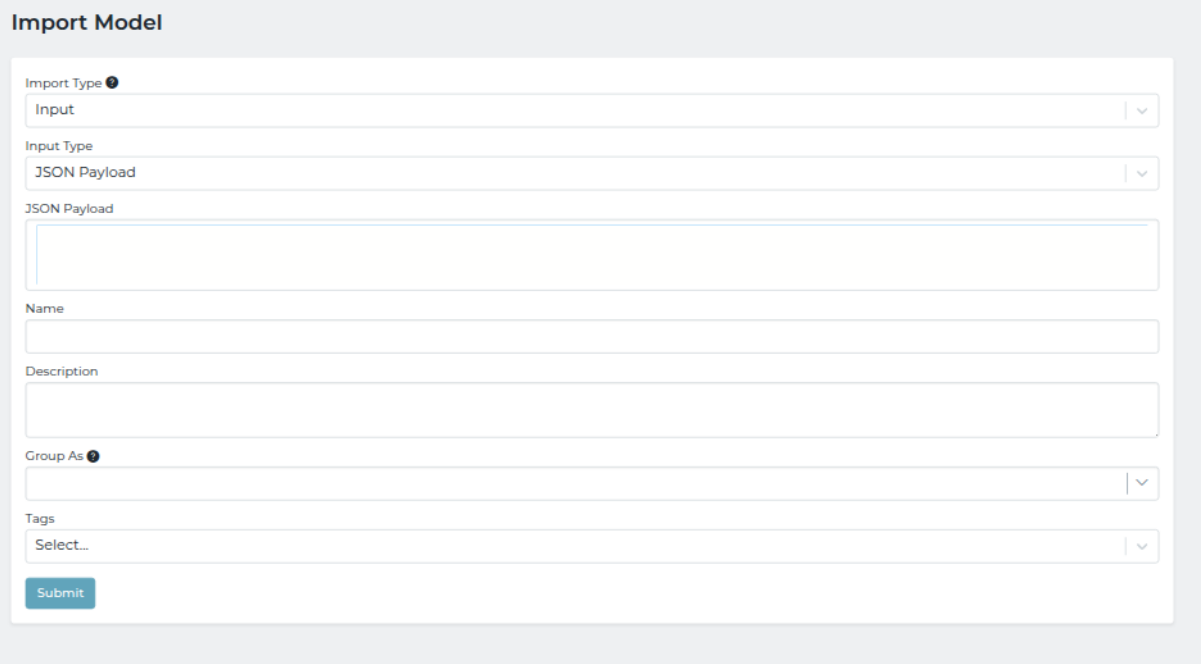
Input Reference

To import from an existing Input Reference, set the `Import Type` to **Input** and `Input Type` to **Input Reference**. Use the Reference Panel to select the desired input, optionally specify other model configuration, and press **Submit**. During import, the specified input reference is read and a new model is created based on the data returned by the input. The input must return an object in order for a model to be created. The **Path** may be used to a specific element of the data returned by the input.



JSON Payload

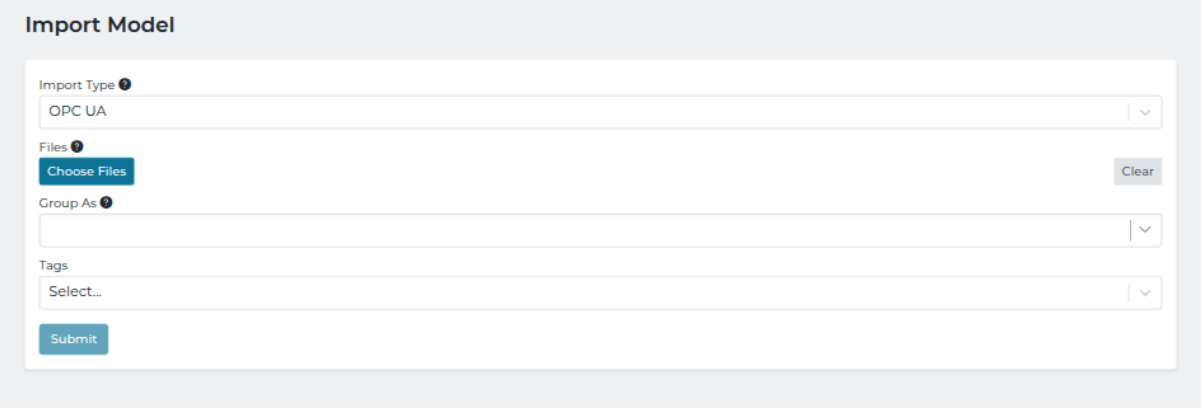
To import from a custom JSON payload, set the `Import Type` to **Input** and `Input Type` to **JSON Payload**. Paste the JSON payload, optionally specify other model configuration, and press **Submit**. A new model will be created using the specified JSON payload. Only objects are supported.



OPC UA Node Set

To import using OPC UA Node Set XML, set the `Import Type` to **OPC UA**. Use **Choose Files** to select the Node Set XML files that define the OPC UA Information Models (and dependencies), optionally specify other model configuration, and press **Submit**.

During import from an OPC UA Node Set, a Model will be created for all Object and Variable types defined in the Node Set XML files.



Models | User Guide

Source: https://guide.highbyte.com/configuration/model/models/

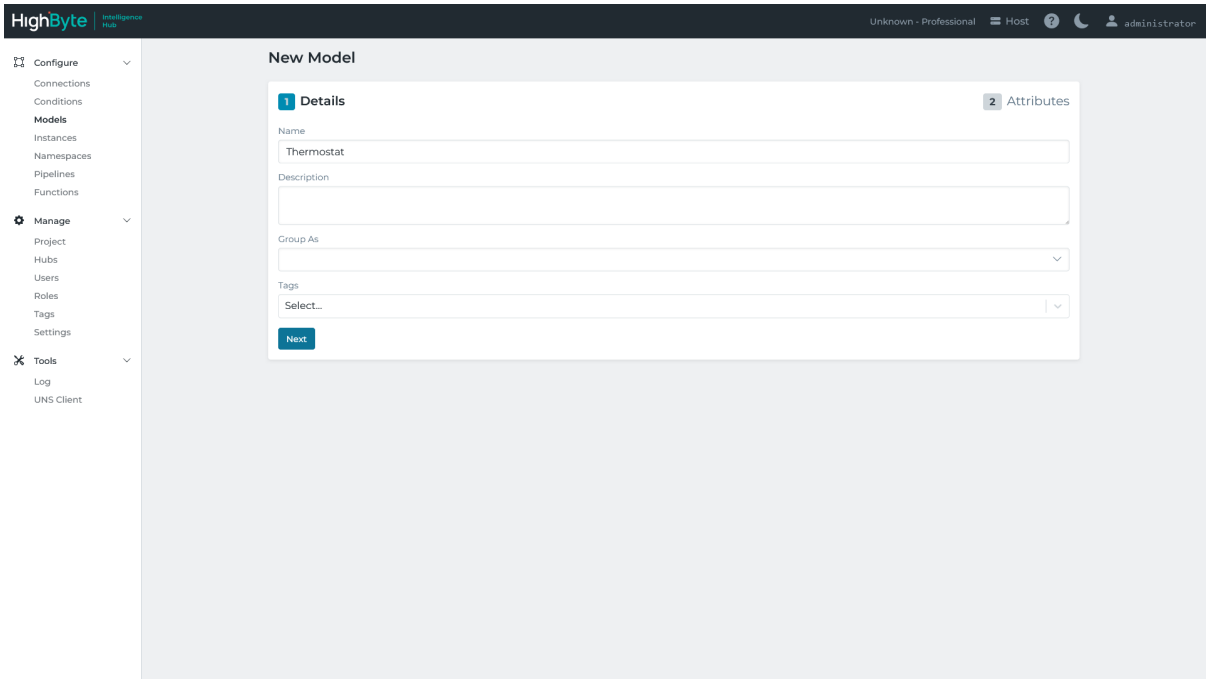
Models

Models provide a standard definition for an asset, process, product, system, etc. A model consists of a collection of attributes that have a unique name and data type. For example, a model for a Motor might have on/off state as a boolean, speed as a float, and location as a string.

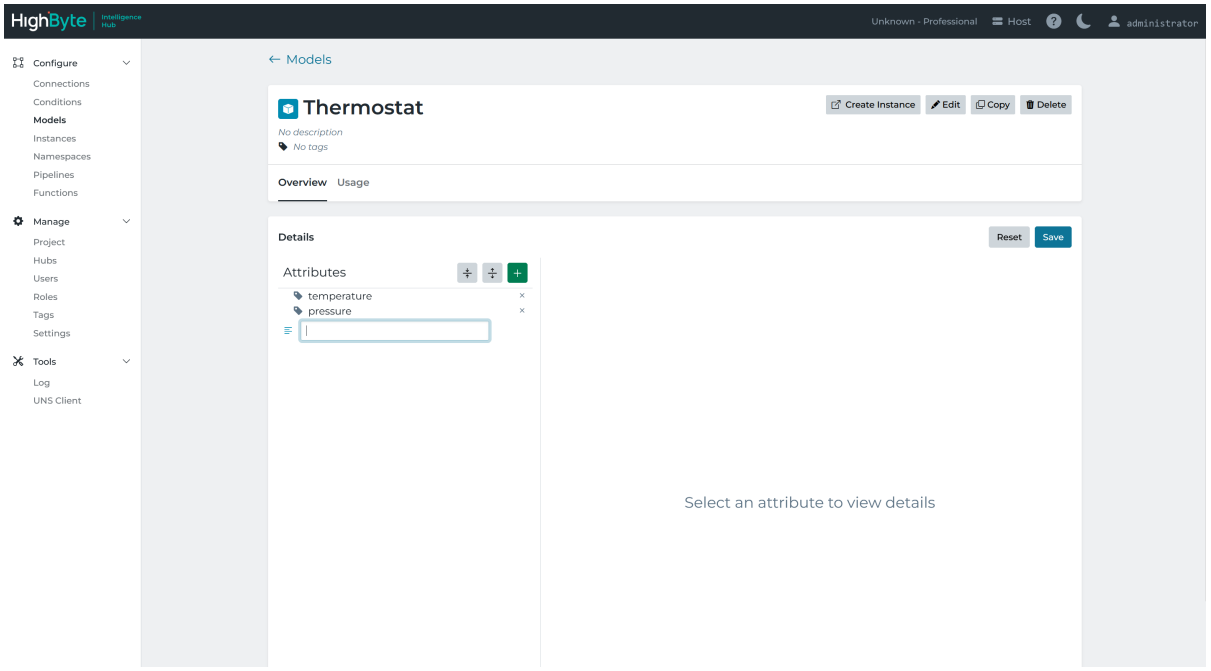
Models abstract away the specifics of the data source. Once modeled, it doesn't matter if the motor comes from vendor A or B, the data for the motor looks the same and contains all the context required for the use case.

Create a Model

1. Navigate to *Models* in the configuration's Main Menu. Here you can add, edit, and view models. To add a model, click the **New Model** button.
2. Enter a *Name* of the model, followed by optional *Description* and a *Group As* folder to categorize the model. Click *Next* to continue.



3. Click *New Attribute* and set the attribute's *Name* and *Type*. Additional attribute properties are described below. Continue to add additional attributes that should be assigned to this model and click the *Submit* button when finished.

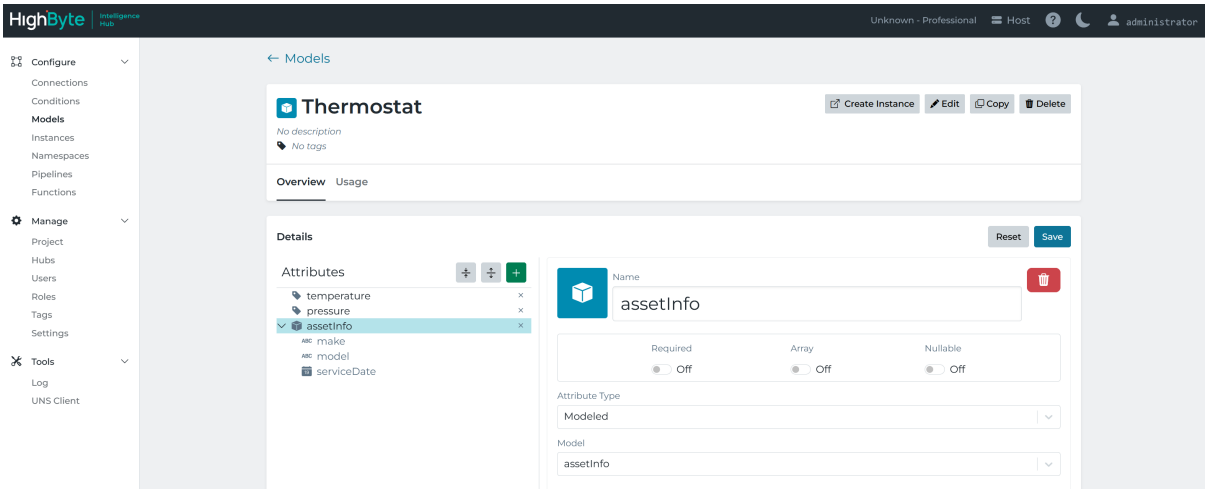


Attributes have the following settings.

Setting	Description
Name	The name for the attribute to be included in the model.
Attribute Type	Attributes can be Simple (string, numeric, etc), Modeled (child Model definition), or Object (hierarchy). Object types mean the model contains inline hierarchy. Simple and Modeled types have additional configuration described below.
Simple Type	Only shown if the Attribute Type is Simple. Select the primitive type (ex. Int, Real32, String, File, etc).
Model	Only shown if the Attribute Type is Modeled. Select the child model to reference. Use this to compose model definitions using sub models.
Array	Specifies if the attribute is an array of values of the specified type.
Required	Specifies if an instance of this model must populate the attribute. Only required attributes will generate errors when they cannot be populated.
Nullable	If enabled the attribute can output a value of “attributeName”: null in the payload
Description	Optional description of the field.
Default Value	Default value to use for the attribute if the Instance does not override the value. The value is expressed in JSON form. Strings must be quoted.

Model Composition

When setting an attribute to a Modeled type, the parent Model is now composed of the child model. This is shown below where the Thermostat model has an attribute that is an assetInfo model. The assetInfo model attributes are shown in the tree, but they are read only. When creating an Instance based on the parent model, the Instance contains all the attributes from the parent and child models.


HighByte
Intelligence Hub
Unknown - Professional
Host
administrator
Configure
Connections
Conditions
Models
Instances
Namespaces
Pipelines
Functions
Manage
Project
Hubs
Users
Roles
Tags
Settings
Tools
Log
UNS Client
← Models
Thermostat
Create Instance
Edit
Copy
Delete
No description
No tags
Overview
Usage
Details
Reset
Save
Attributes
temperature
pressure
assetInfo
make
model
serviceDate
Name
assetInfo
Required
Array
Nullable
Off
Off
Off
Attribute Type
Modeled
Model
assetInfo

Namespace | User Guide

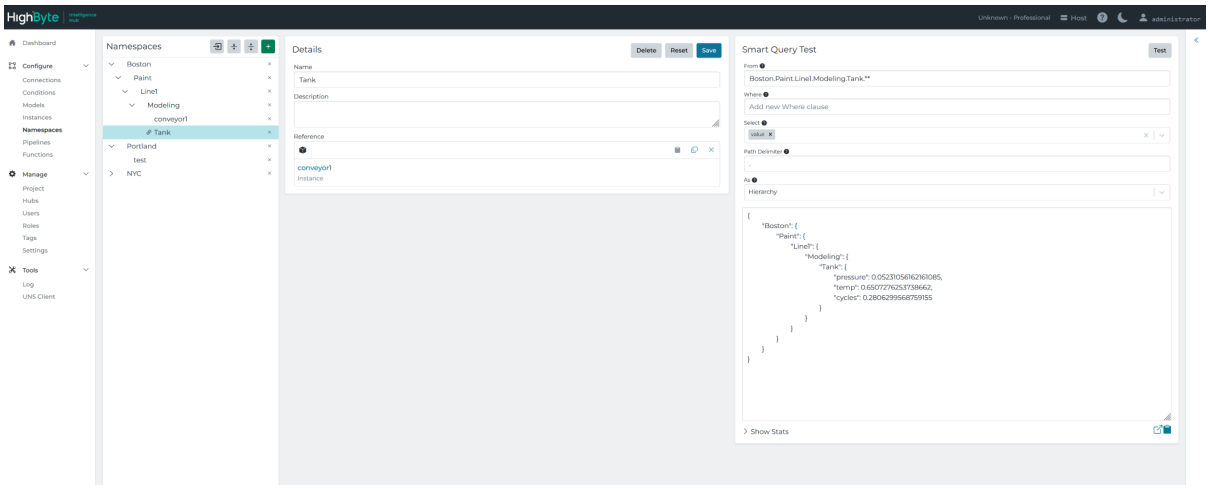
> Source: https://guide.highbyte.com/configuration/namespace/

Namespace

A namespace organizes sources of data into a logical hierarchy. A common use of namespaces is to organize an ISA-95 hierarchy of sites, areas, and lines and then populate this with modeled data like motors, machines, and other assets.

Once a namespace is built the Smart Query Stage in pipelines is used to query the namespace, either in logical parts or as a whole. For example, a smart query can read all the motors in the namespace and publish the data to MQTT using the namespace hierarchy as the topic path.

Creating a namespace



1. Navigate to Namespaces
    - Click *Namespaces* in the configuration's Main Menu to open up the namespace
    - There is only one namespace but the namespace can have multiple root nodes
2. Add Namespace Nodes
    - Click the + icon at the top of the tree.
    - Enter the name of the node in the tree and hit *Enter* to add the node.
    - Continue to add and name nodes, either at the root of the tree or by building hierarchy
    - *Note: that nodes are saved as they are added*
3. Add a source of data to the namespace *(Currently Inputs, Instances, and Callable Pipelines are supported)*
    - **Drag and Drop** - Sources can be dragged from the reference panel on the right
        - Drag a source on to the `Drop Reference` area of an existing node to update it's source
        - Drag a source in to the tree to create a new node with that source at that position
4. Reorder the namespace
    - Nodes can be reordered within the namespace by dragging the node to a new position

References

Each namespace can reference one source. For more information on using References see: References

A warning may appear for some references created in 4.1 or prior, these reference will need to be manually migrated. For more information see: Migrating Namespace Node Parameters

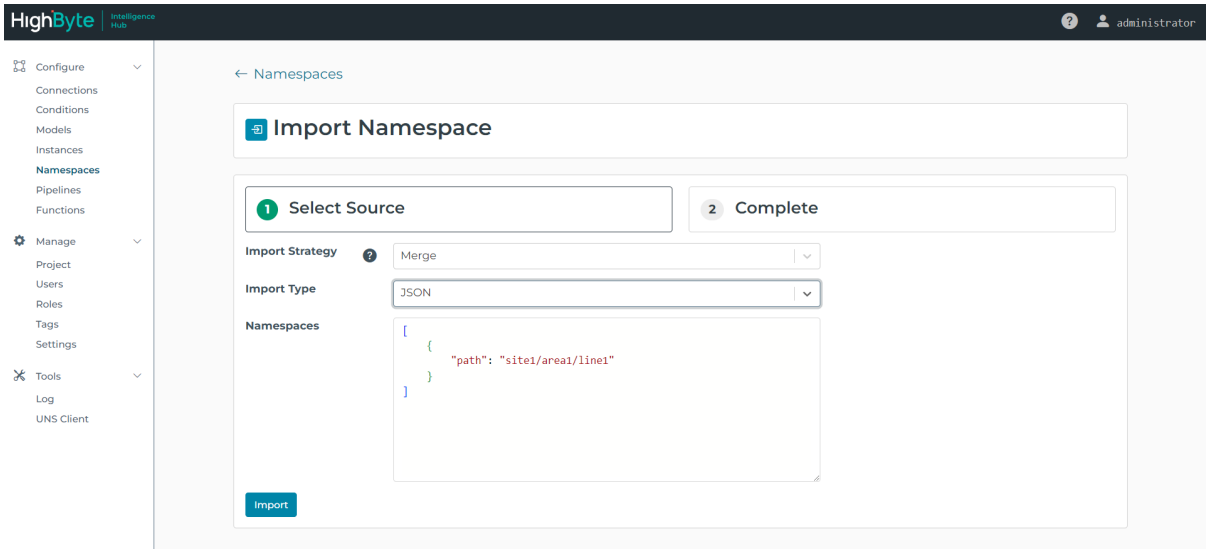
Test reading a namespace

To test read the namespace, select the *Read Node* at the bottom. This reads the full namespace and returns the result as JSON.

Importing a Namespace

To quickly build a namespace, select the `Import` icon above the Namespace tree. Supported import formats include CSV and JSON. Merge makes the import additive, and Replace deletes the existing namespace and replaces it with the imported one.

Note imports do not support linking sources to a node. If Merge is used, and the node already exists, the reference/source configuration is left as is.



CSV Example

```
path
site1/area1/line1
site/area1/line2
```

JSON Format

```
[
    {
        "path": "site1/area1/line1"
    },
    {
        "path": "site1/area1/line2"
    }
]
```

Smart Query

Smart queries allow you to dynamically query the namespace to read only the data you need. See the Smart Query Stage for details on the smart query stage.

Migrate Parameters | User Guide

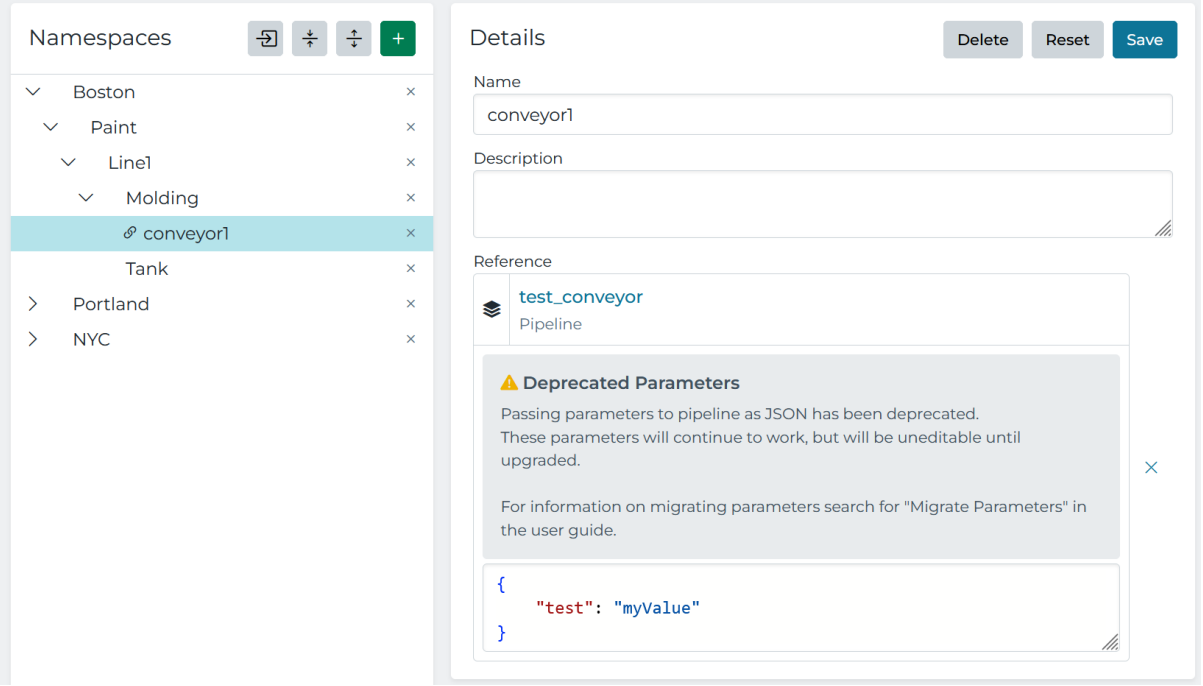
Source: https://guide.highbyte.com/configuration/namespace/migrate_parameters/

Migrate Parameters

In 4.1 and older versions, namespace node parameters were passed as JSON. When upgrading to 4.2 or newer versions, the parameter syntax is now better defined and easier to use. In cases where you were passing parameters to a namespace node that referenced a Pipeline, the 4.1 syntax for will continue to work, but you'll need to manually migrate the syntax to use the new format. Below is a step-by-step guide on how to perform this migration.

Identify Nodes that Need Migration

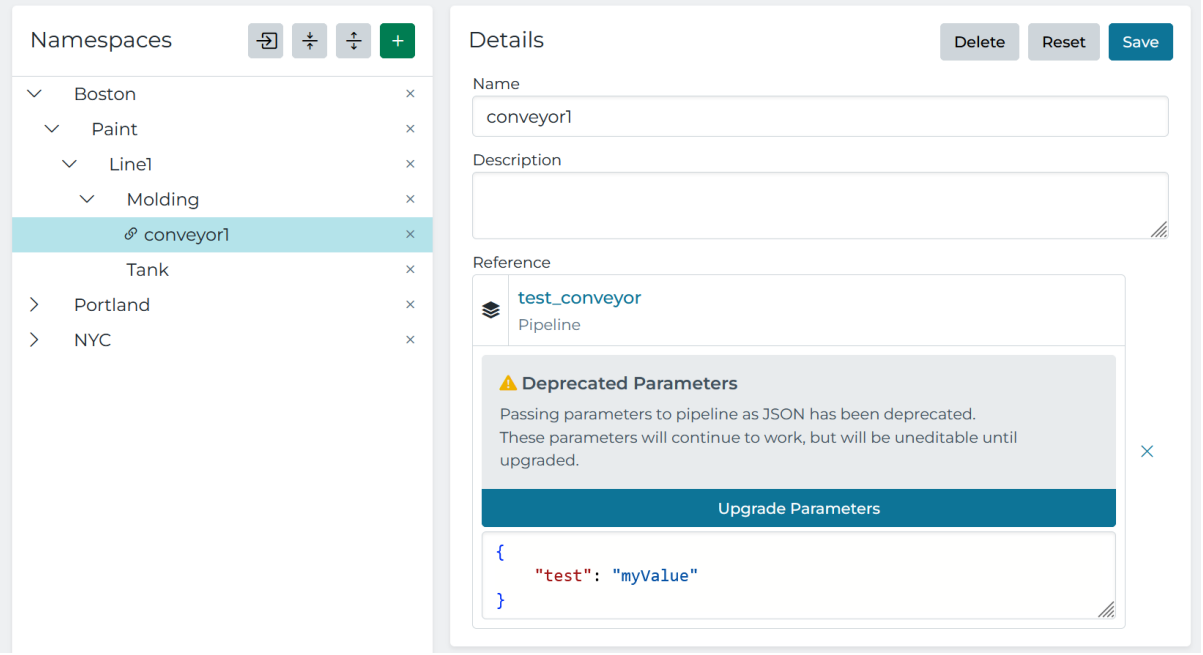
If a node is using the old syntax, you'll see this warning in the UI when selecting the node.



How to Migrate the Syntax

Follow these steps to migrate to the new parameters' syntax.

1. Go to the Pipeline and create a Callable Trigger.
2. Enable the trigger and add the parameters in the trigger settings.
3. Go back to the namespace node, you should now be able to "upgrade" the parameters *(see image below)*
4. You should now see the parameters you defined on the trigger, and can fill them in accordingly
5. Save, and perform a smart query test on the node to ensure it's working as expected


Namespaces
Boston
Paint
Line1
Molding
conveyor1
Tank
Portland
NYC
Details
Delete
Reset
Save
Name
conveyor1
Description
Reference
test_conveyor
Pipeline
Deprecated Parameters
Passing parameters to pipeline as JSON has been deprecated.
These parameters will continue to work, but will be uneditable until upgraded.
Upgrade Parameters
{
"test": "myValue"
}

Other | User Guide

> Source: https://guide.highbyte.com/configuration/other/

Other

- References
- Parameters
- Templating
- Functions
- Output Templates

Functions | User Guide

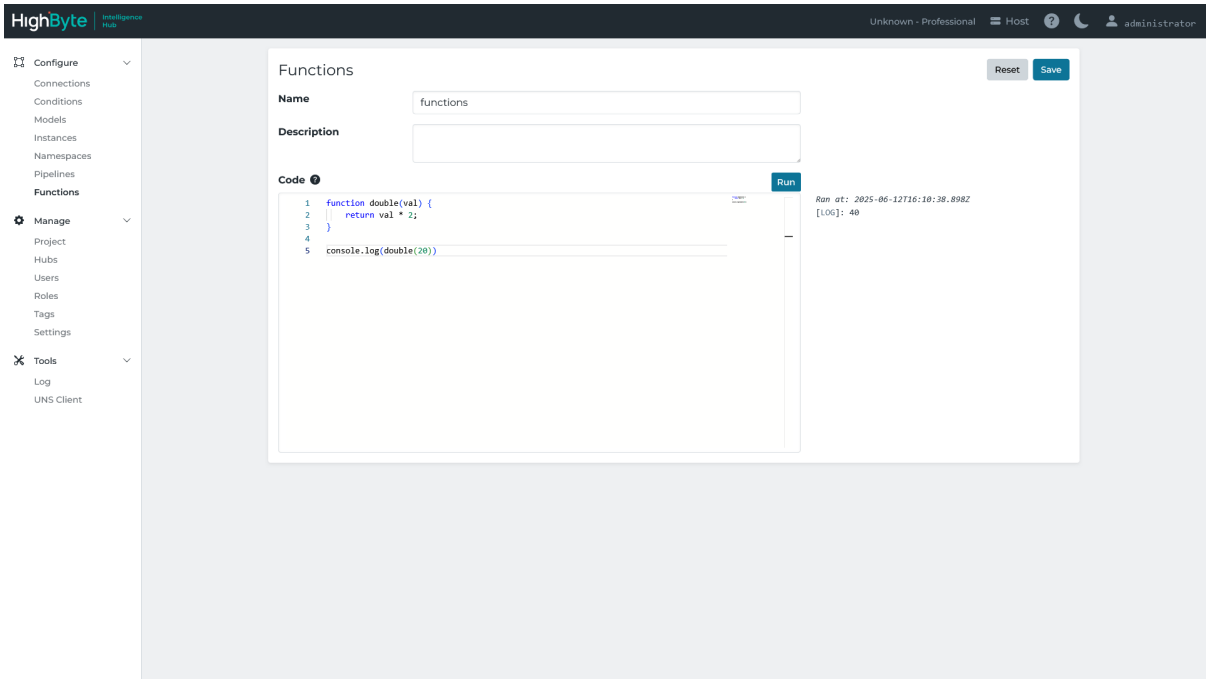
> Source: https://guide.highbyte.com/configuration/other/functions/

Functions

In some cases, there are snippets of JavaScript code that get reused in many expressions across many Instances. This can be code that checks for a certain set of values or performs other logic. Functions are a way to write these once, and reuse them anywhere in the product that supports JavaScript, including expressions, triggers, and conditions.

To access Functions, navigate to *Functions* in the configuration's Main Menu. Enter the JavaScript in the *Code* section. Functions can be tested by selecting *Run*. This executes the JavaScript in the browser and outputs the results to the right of the Code panel.

Often to test a function, you must write some test code to call the function and log output results to the console. Keep in mind when saving these changes that functions are instantly available to all running data flows. Only Save once you have confirmed the functions work as expected.



Expression Imports

The Intelligence Hub supports importing npm-compatible CommonJS modules and using them in expressions. For example, there may be a parsing library required to parse a custom formatted payload.

To use this feature 3rd-party modules must:

1. Enable `allowExpressionImports` in the Application Settings. This creates a `expression-imports` directory of the hub's `directories.appData` directory
2. Use the `npm install` command from the `expression-imports` directory to install 3rd-party modules for use by the hub.
3. These imports are now available in JavaScript expressions

NOTE: CommonJS packages that rely on built-in Node.js core Modules or native NPM modules are not supported.

Output Templates | User Guide

> Source: https://guide.highbyte.com/configuration/other/output_templates/

Output Templates

Some outputs support a special Template syntax to format the payload. This can be useful when the output format needs to be finally controlled and it can't be accomplished in pipelines. For example, if the endpoint expects a special SOAP or XML format.

The 'Template' setting for these Outputs uses Apache FreeMarker to provide this control.

Apache FreeMarker Special Values

- `${value}`
  - The value being written out. By itself this evaluates to the default JSON output
- `${values}`
  - Used in cases where the output is an array of values versus a single object. Like `${value}` this by default evaluates to the default JSON output
- `${value.attribute_name}`
  - Index into a model to get the value for `attribute_name`. For example, a model with a cycleCount attribute would be referenced as `${value.cycleCount}`
- `${value.name}`
  - Returns the name of the model instance or alias
- `${value.quality}`
  - Returns the quality as a string, either Good or Bad
- `${value.time}`
  - Returns the timestamp as an epoch (UTC)
- `${value.type}`
  - Returns the model name
- `${value.elements}`
  - Returns a list of key value pairs, where key is the attribute name and value is the attribute value
- `${value.array}`
  - Returns an array required by FreeMarker array operations
- `${value.rawString}`
  - For use with file typed values to access the raw string data contained in the file

Note: The syntax can be chained together. So if the model has a child model called "motor" with an attribute "amps", you can access amps using `${value.motor.amps}`.

Examples

With this API, here are some example use cases. Please see the FreeMarker documentation online for FreeMarker specific syntax.

Decorate payload (add around payload)

```
{
  "index": "myindex",
  "source": "mysource",
  "payload": [
    <#list values as value>
    ${value}<#if value_has_next>,</#if>
    </#list>
  ]
}
```

Rename a field

```
{

  "payload": [

  <#list values as value>

    ${value?replace("_name", "name")?replace("_model","model")?replace("_timestamp", "timest
    <#ifvalue_has_next>,</#if>

  </#list>

  ]

}
```

Xml

```
<root>
  <values>
    <#list values as value>
      <value>
        <#list value.elements as name, v>
          <#if name != "motor">
            <${name}>${v.string}</${name}>
          </#if>
        </#list>
        <motor>
          <#list value.motor.elements as name, v>
            <${name}>${v.string}</${name}>
          </#list>
        </motor>
      </value>
    </#list>
  </values>
```

```
</root>
```

Values in OPC format

```
{
  "timestamp": ${.now?long},
  "values": [
    <#list values as value>
    {
    "id": "${value.name}",
    "v": ${value},
    "q": ${value.quality},
    "t": ${value.time}
    }<#if value_has_next>,</#if>
    </#list>
  ]
}
```

Values in influxdb format

```
${value.type},name=${value.name} <#list value.elements as name,v>${name}=${v}<#if name_has_n
```

Transform time

```
{
  "values": [
    <#list values as value>
    {
    "id": "${value.name}",
    "v": ${value},
    "q": ${value.quality},
    "t": "${value.time?number_to_datetime?iso_utc}"
    }<#if value_has_next>,</#if>
    </#list>
  ]
}
```

Parameters | User Guide

Source: https://guide.highbyte.com/configuration/other/parameters/

Parameters

Parameters provide a way to create generic sources (Inputs, Instances, and Pipelines) that take arguments to return different results. This is similar to functions in programming that take arguments.

Here are a few examples:

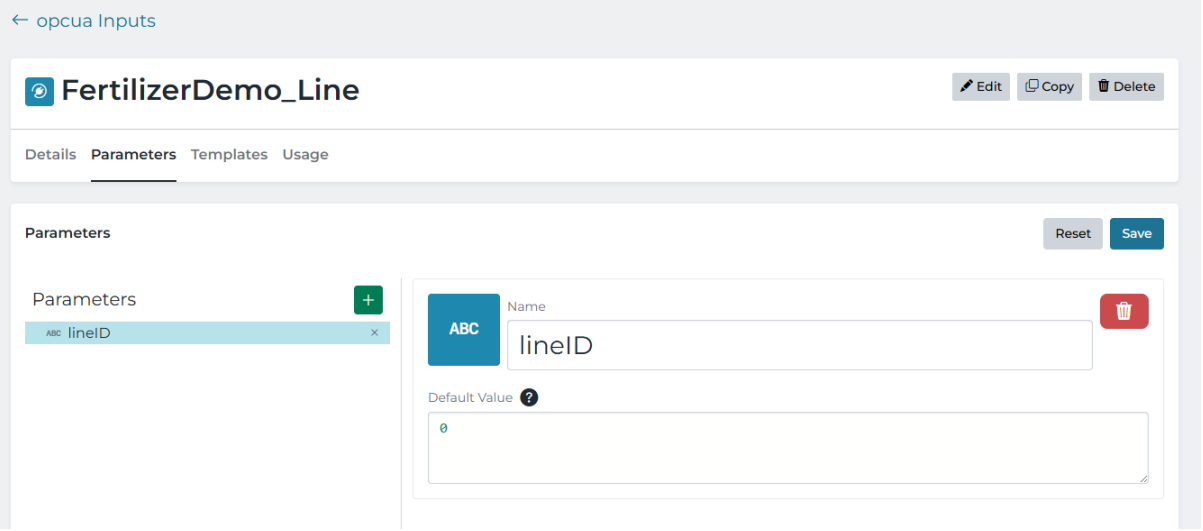
1. An OPC UA tag input that takes a tag address parameter that's used to control the tag that is read
2. A Mixer Instance that takes the mixer identifier which is used to scope a SQL query to return the current batch information for the mixer
3. A getWorkOrder pipeline that takes the line number and returns the current work order for the line

Below are examples of how to use parameters for Inputs, Instances, and Pipelines.

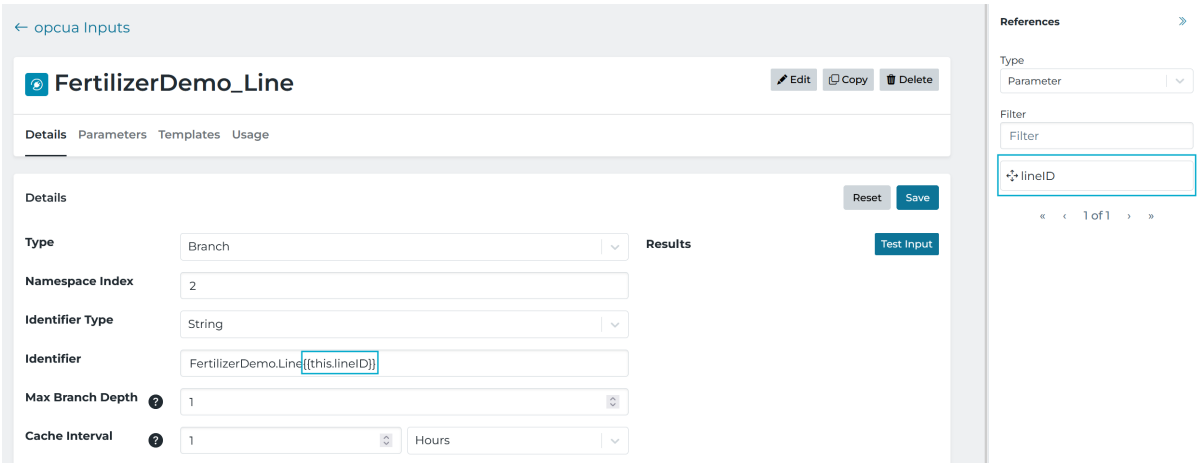
Input & Instance Parameters

Input and Instance Parameters are configured in the same way. Navigate to the details page of the Input or Instance and select the `Parameters` tab. This launches a UI that allows you to define the parameter names and default values.

In this example a single parameter named lineID is defined, with a default value of zero. Parameters are always a flat list of attributes, and for Inputs and Instances they are string types.

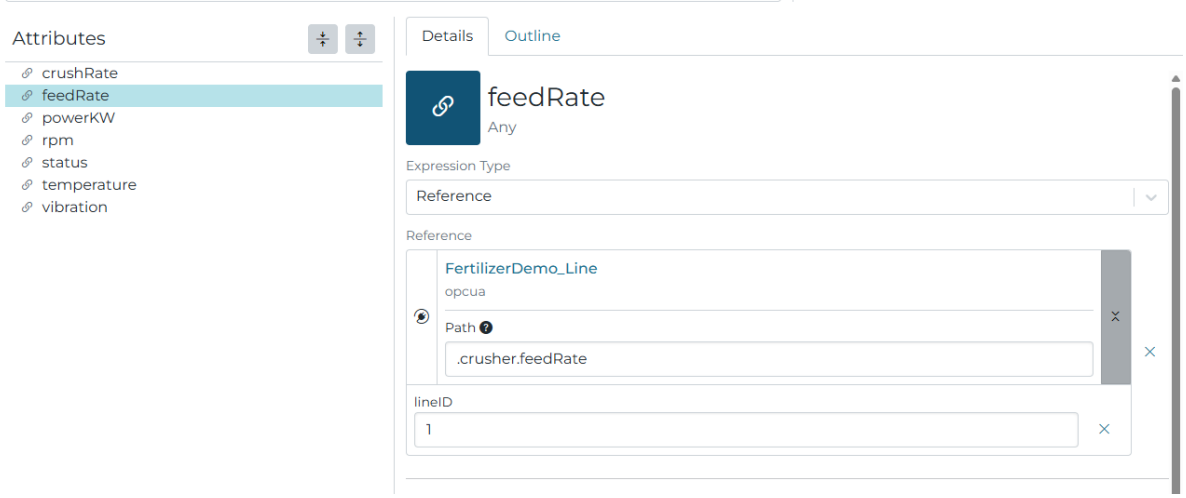


Once configured, navigate back to the details view and you'll see that the defined parameters are in the Reference Panel on the right. You can now use this parameter in the Input or Instance by dragging and dropping the parameter from the reference panel into a field, or by using the `{{this.parameterName}}` syntax. In the below example the lineID is used to construct the tag identifier.

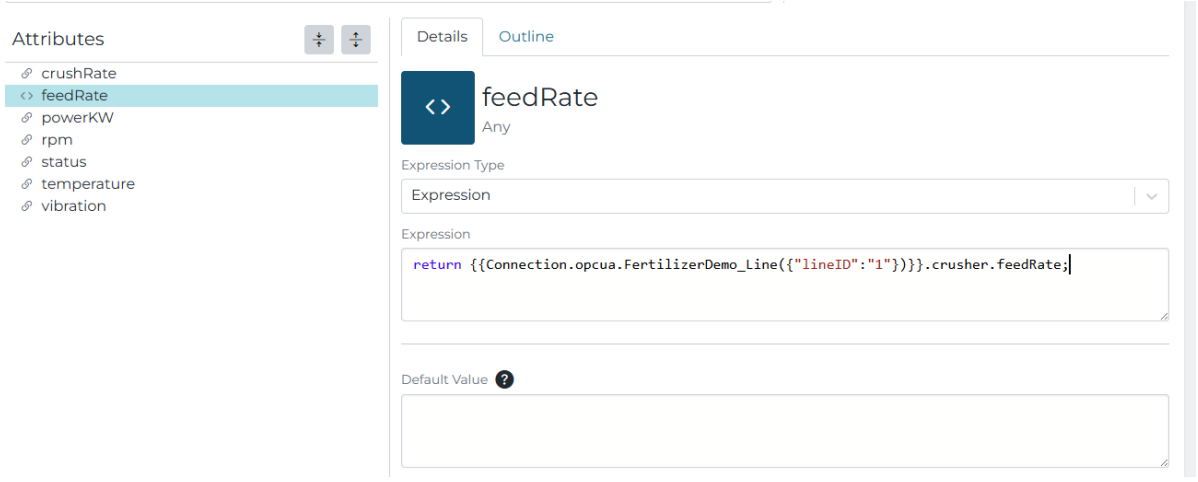


Now when referencing this Input in an Instance or Pipeline, you'll see the parameters and be able to fill in the values.

Here is an example of using this Input in an Instance attribute reference.

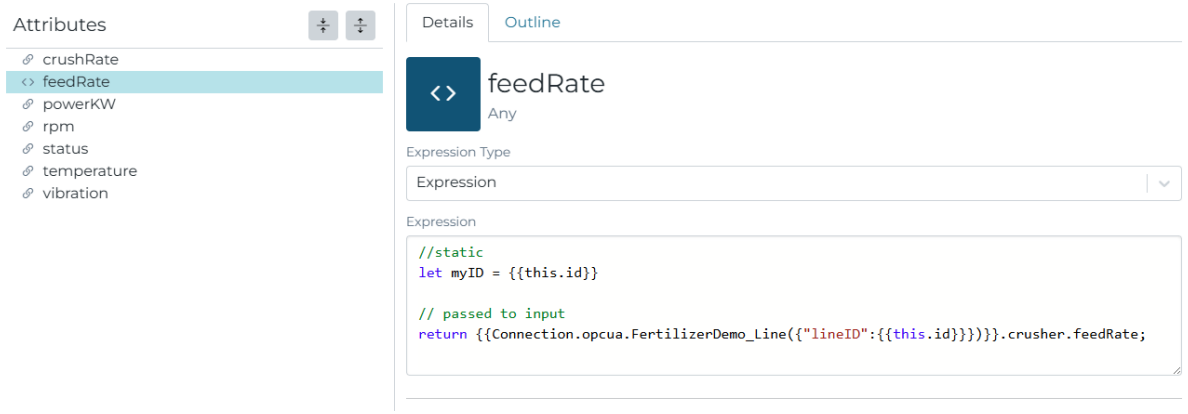


Here is an example of using this Input in an Instance attribute expression. Note expressions take parameters in JSON format.



Instance Parameters

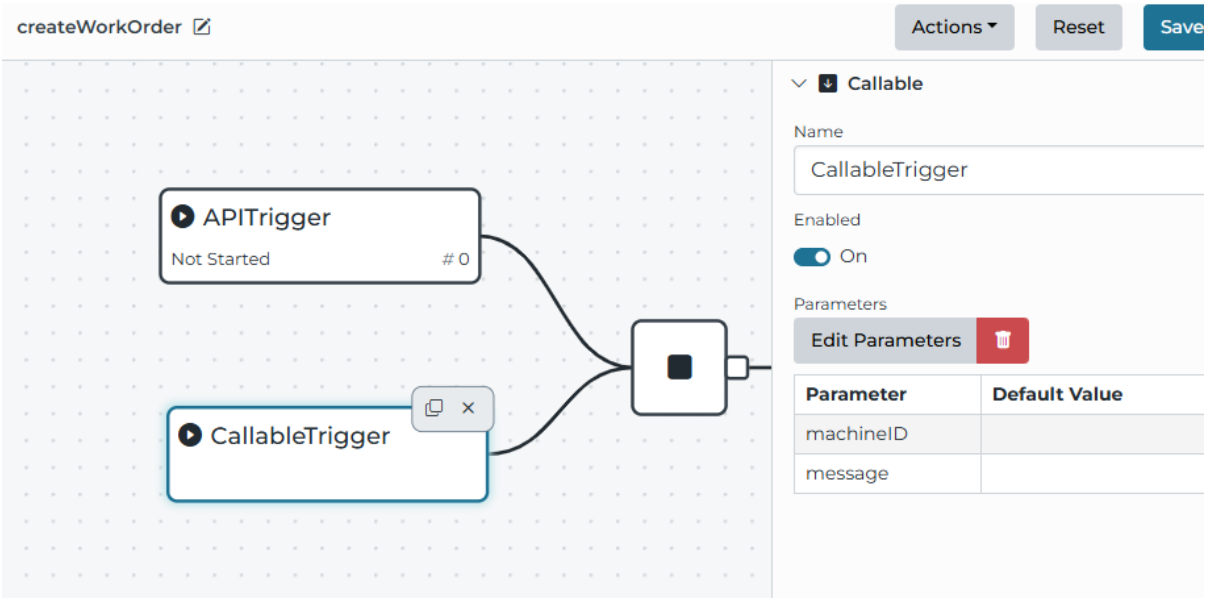
Instance parameters work the same as Input parameters. Below is an example of using a parameter in an attribute expression field; both as a static value and passing it to a source. Note the expression fields replace the `{{this.param}}` syntax before executing JavaScript.



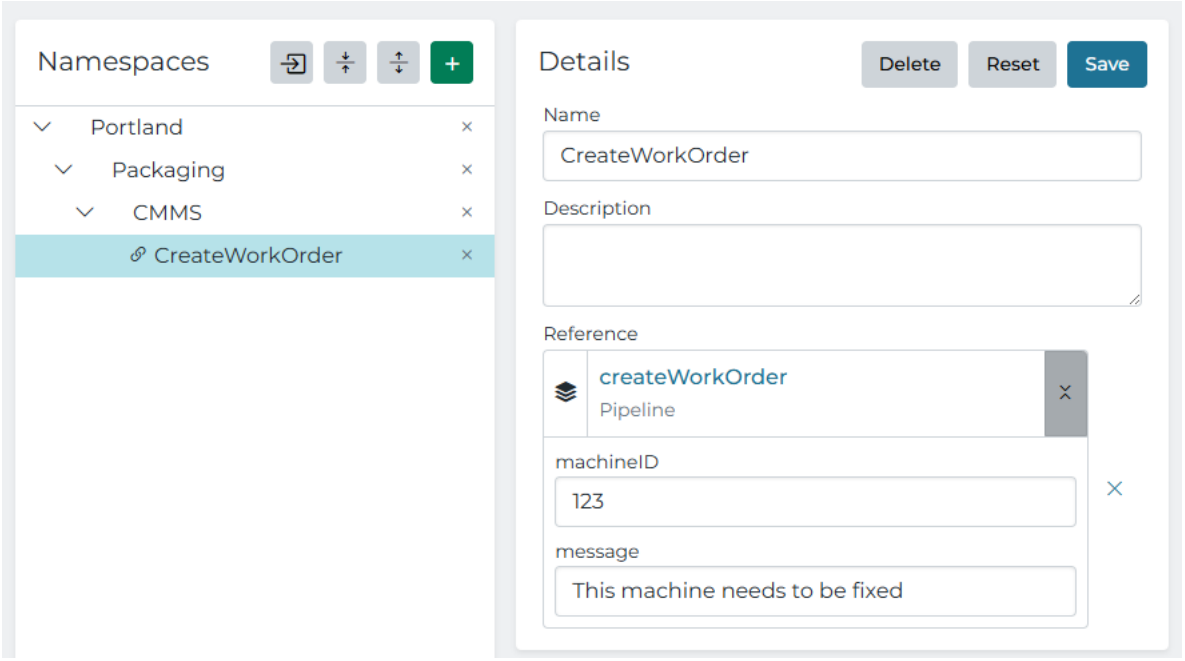
Pipeline Trigger Parameters

Pipelines can take Parameters through Callable or API Triggers. In both cases the Parameters are defined on the trigger similar to how they are configured for Inputs and Instances. The biggest difference is trigger parameters can set a data type.

Below is an example of a Callable Trigger that takes two arguments; machineID and a message to create a workorder.



Once defined, Callable Trigger parameters are exposed to Namespace nodes that reference the Pipeline. Below is an example.



Parameter Data Types & Default Values

Parameter data types are enforced, and if a call is made with an invalid data type the read will fail. If a parameter does not have a default value, and no value is provided during the read, the read will also fail.

References | User Guide

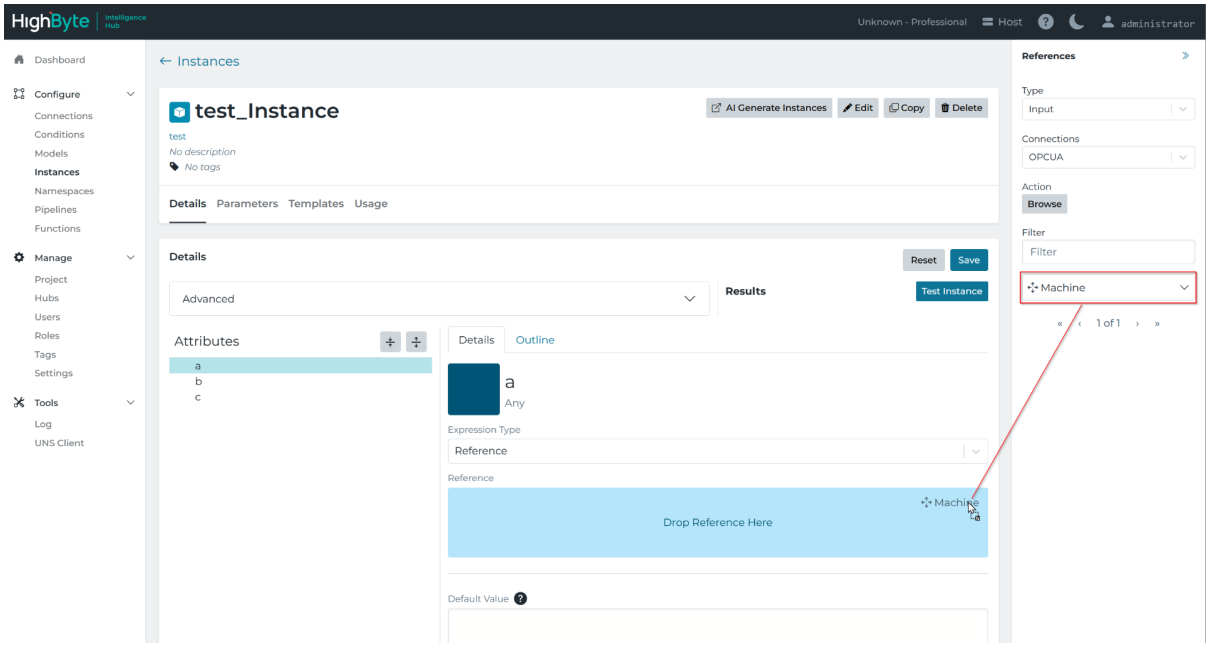
> Source: https://guide.highbyte.com/configuration/other/references/

References

References are a powerful way to access data sources in the Intelligence Hub.

Populating

To populate a Reference simply drag and drop references from the Reference Panel on the right side of the screen.



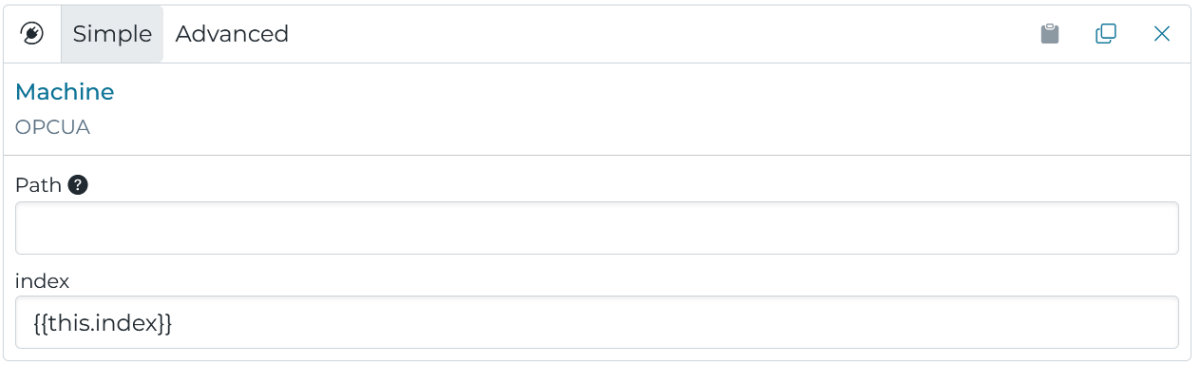
Modes

Simple

By default, References are set to "Simple" mode.

In Simple mode the Reference Name will hotlink (if applicable) to that element, and a list of Parameters will auto-populate if that reference supports parameters.

In this example, this reference is to a Machine Input on an OPCUA connection. Since that connection has a `index` parameter that field is shown below the path, and in this case, an index is being dynamically provided using a parameter in an Instance.



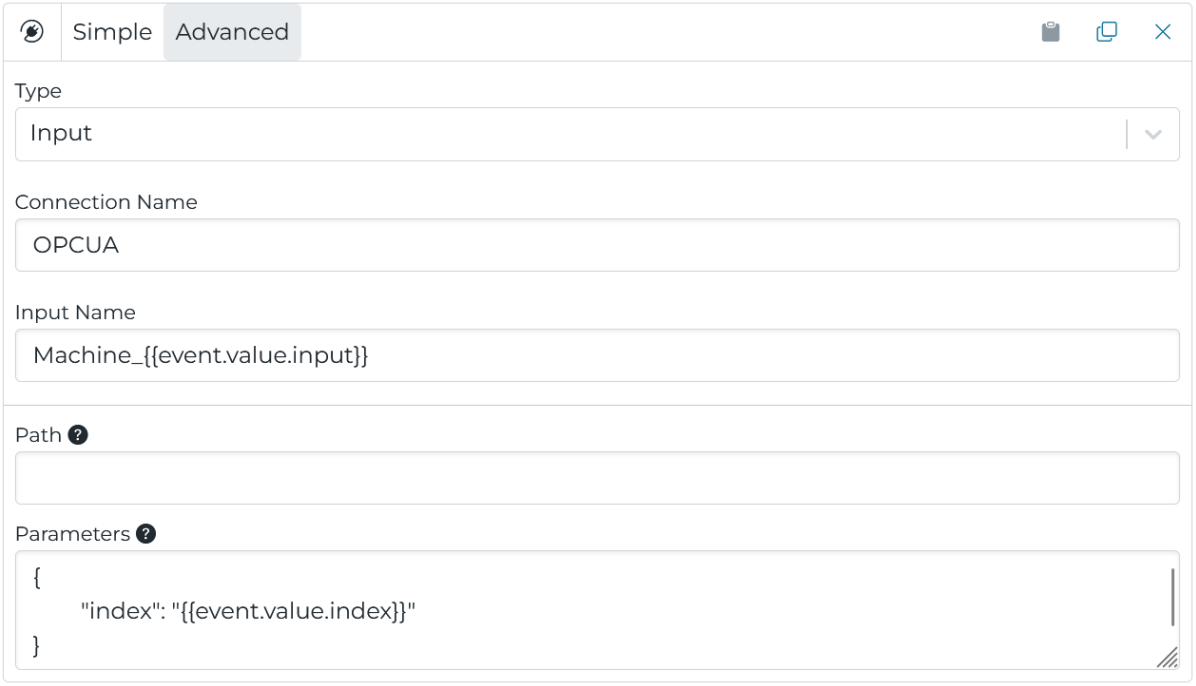
Advanced

A Reference can be changed to "Advanced" mode by clicking the "Advanced" button in the top left corner.

> A Reference can be returned to Simple mode by clicking the "Simple" button, but the reference must be in a valid state. (dynamic names are not supported in Simple mode)

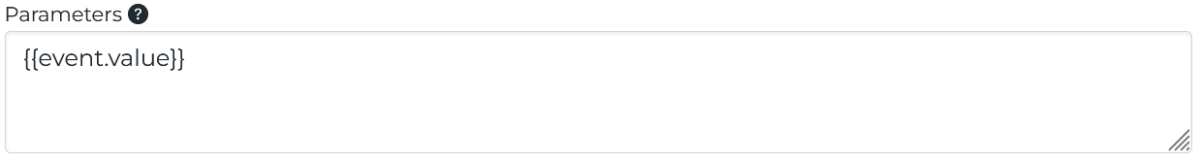
In Advanced mode references can by dynamically driven by parameters, event values, and system variables.

In this example, this reference is to a Machine Input on an OPCUA connection, and the Input name for the Machine is being dynamically driven by the event value in a Pipeline. The user knows that these Inputs all have an `index` parameter and that parameter is also being populated using the event value from a Pipeline.



Using Event Value as Parameters

Instead of building out an object for Parameters, since `event.value` produces an object, this could also be injected directly in to the parameters field.



Path

A "Path" can be added to extract specific data from the data source. For example:

- If the data source produces an array, a path of `[0]` would return the first element from that data source.
- If the data source produces an object with the property "temperature", then a path of `.temperature` would return the value of that property
- If the data source produces an arrays and objects then the path can be a combination of the two. For instance `[0].temperature` would return the temperature property of the first element in the array.

Copy/Paste

Reference can be Copied using the copy button in the top right of the reference.

If a Reference is copied it will be remembered inside of the current browser tab, and can be pasted in to another Reference field anywhere in the Intelligence Hub.

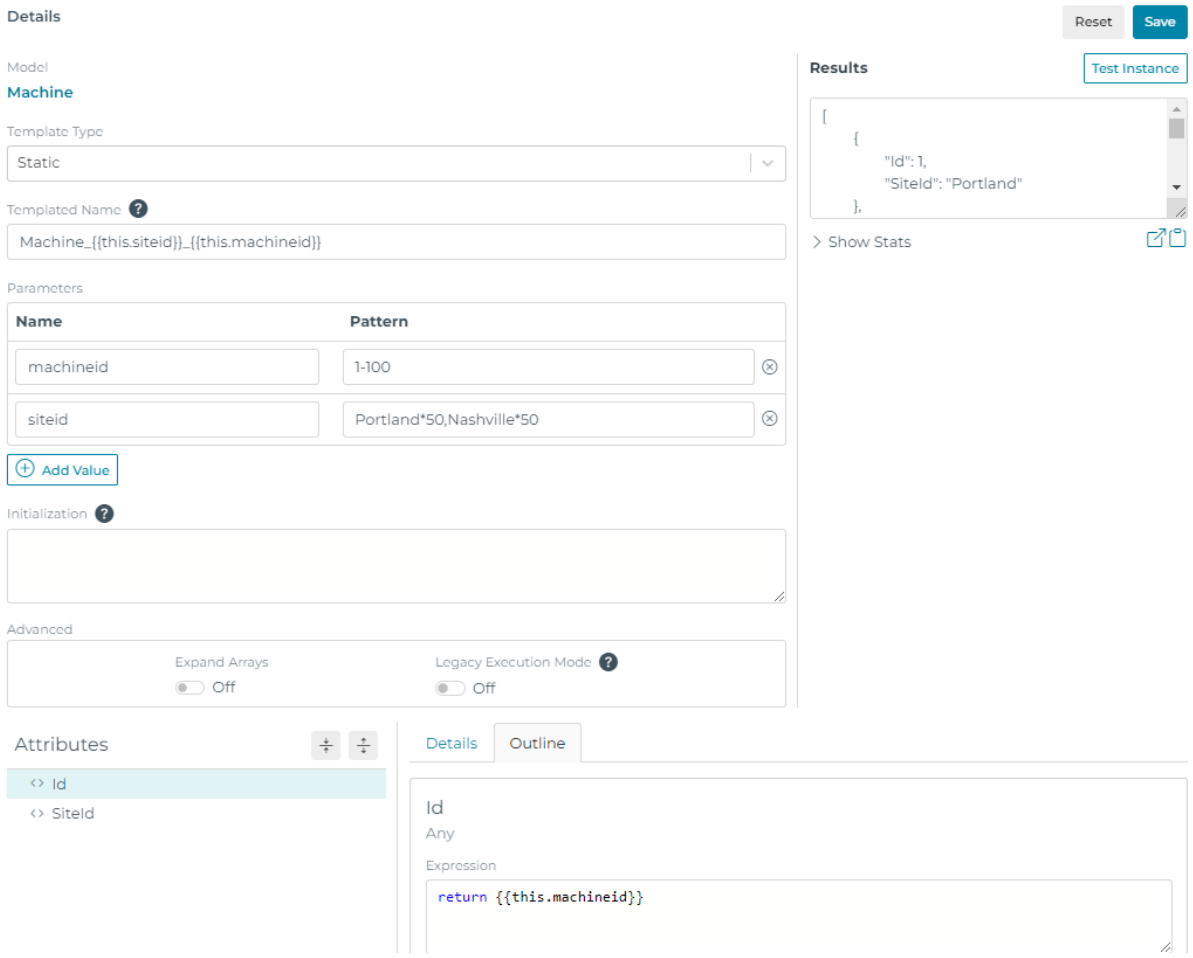
> Paste will not work if the reference type is not supported in that location.
> For example, Namespaces do not support Advanced reference.

Templating | User Guide

Source: https://guide.highbyte.com/configuration/other/template/

Templating

Templating is supported for Inputs, Instances, and Flow Triggers in Pipelines. Templating is a quick way to create 100's of configuration objects when each object differs by only a small amount. For example, if an OPC UA Server has 100 machines and the addressing of each machine only differs by a unique machineID, a single Input can represent all 100 machines.



Template Settings

The template settings for Inputs, Instances, and Flow Triggers are described below. Note that templates can be statically defined or dynamically defined. When dynamic templates are used the template definitions come from an external source like a database or CSV file.

Setting	Description
Use Template?	Disabled by default. When enabled, the object uses it's template configuration.
Templated Name	Specifies a name for the templated object. If left blank, the name is taken from the Input, Instance, or Flow Trigger name. When templated objects are expanded, they are given unique names. For example, if the template name is `MyInstance`, the Instance names generated are `MyInstance1`, `MyInstance2`, `...` `MyInstance100`. Name expansion supports padding. If the name is

Setting	Description
	`MyInstance001` the generated names are `MyInstance001, MyInstance002, ... MyInstance100`. Template parameters can be used in the template name for more control over the name. For example, a template name of `MyInstance{{this.nameid}}` is replaced with the `nameid` parameter.
Type	Static or Dynamic. Static templates are hardcoded and fixed (see parameters below). Dynamic templates are read from an external source like a SQL database or CSV file.
Parameters	Parameters are used for Static template types only. Template Parameters control what is replaced in the configuration. Each parameter has a name and a range of values. As an example, the following replaces machineID with the values 1, 2 and 3. machineid: 1,2,3 Each parameter is flexible in terms of the range of values it accepts. See below options for template param formats.
Reference	Used for Dynamic template types only. This is the reference which is read to return the template settings. This can be a SQL input, a Condition, or any other data source. See examples below for the required data shape.

Static Parameter Examples

Below are examples of the different syntax supported for static template parameters.

Example	Description
M1,M2,M3	A discrete list of values: M1, M2, M3
1-100	A range of values, 1 through 100. This is equivalent to typing out 1,2,3,4...100 Numeric padding is supported. 001-100 → 001, 002, ... 100
hello*3	Repeat a pattern N times: hello, hello, hello
"1-AB*3","1-10"	A discrete list of values, using quotes to escape special characters. Note: To escape a quote inside a quoted string, use double quotes. For example """" will produce a value that is a single quote.

The syntax can be chained together to provide maximum flexibility. As an example:

machineid: `1,2,3-5,11*2` transforms into `1, 2, 3, 4, 5, 10, 11, 11`

If multiple parameters are used, each parameter must expand into the same number of values. The first object is created with the first value of each parameter, the second object with the second value, etc. Parameters can be used anywhere in the object configuration.

Parameters are referenced inside the configuration using the {{this.paramName}} syntax.

Dynamic Parameter Examples

The Dynamic Reference option references a source of data. The source must return an array of objects with the same schema, where the keys for each object are the parameter names and the values are the replacement values. Below is an example.

Assume reference `{{Connection.SQL.templateParams}}` returns the following two rows from a SQL table.

```
[
    {
        "param1": 1,
        "param2": 2,
    },
    {
        "param1": 10,
        "param2": 20
    }
]
```

This is the same as assigning static parameters as follows.

`param1: 1,10`

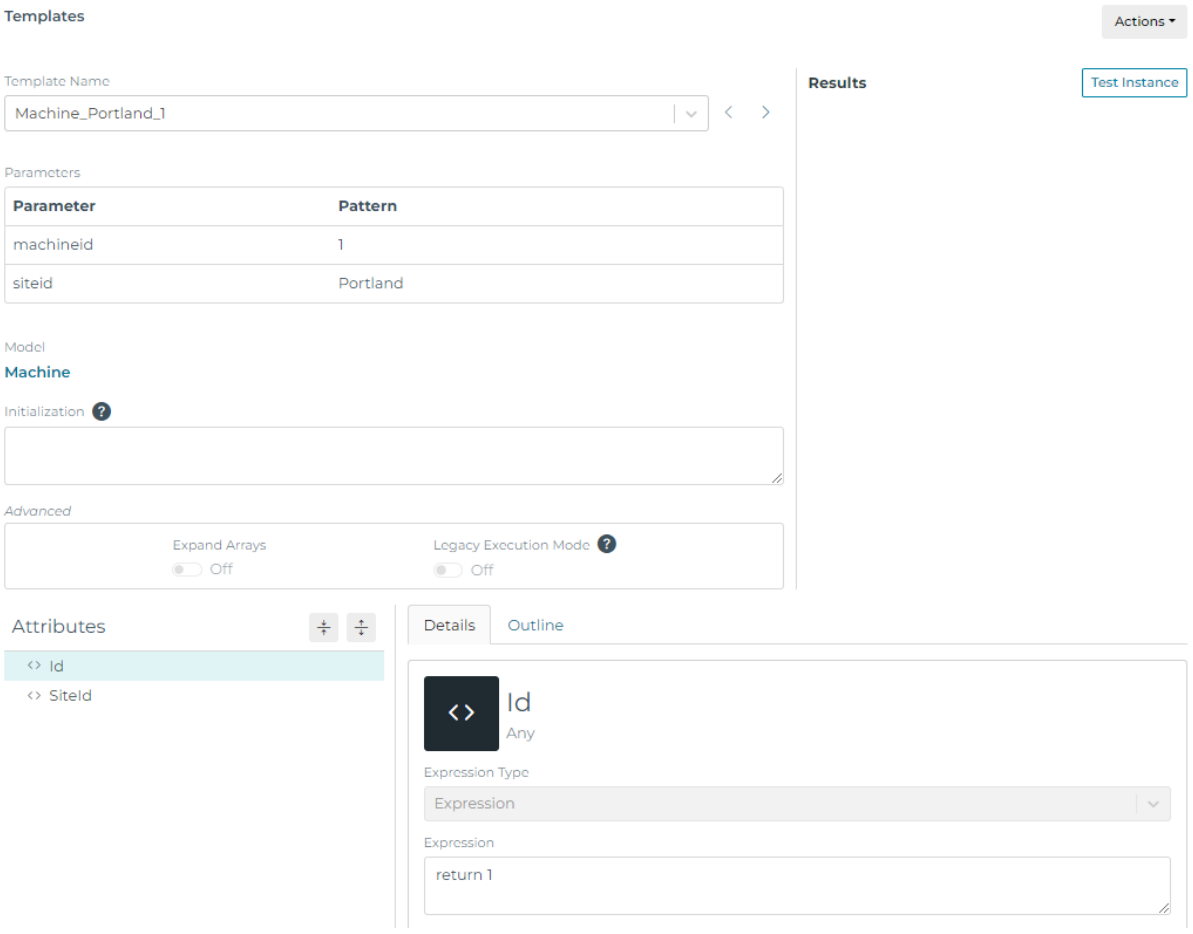
`param2: 2,20`

The Dynamic reference is re-read any time the object definition is saved, or when the flow that references it is restarted. If the raw input data isn't in the above format, use a Custom Condition to transform the input into the required format.

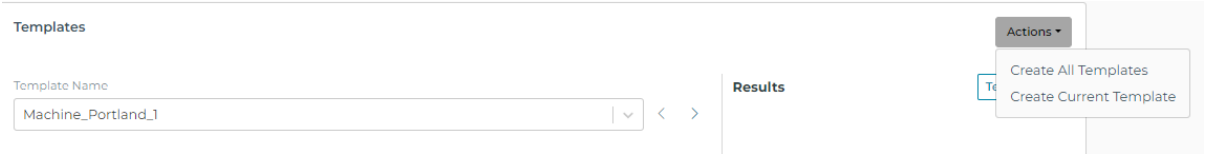
Viewing Templates

To view the generated templates use the Templates navigation option that can be found on the Detail page.

The Template view allows you to select one of the projected templates, view its parameters and resulting configuration, and perform a test read.



Templated objects can easily be converted into concrete objects using the *Actions* menu to either create the current object or create all objects. This allows the use of templating to mass create configuration objects. Once created, the resulting Input or Instance is no longer templated, but contains the name and configuration generated by its parameters.



Templated Instances and Inputs

When a templated Instance is referenced as a source, it expands using the template configuration. As an example, if a Flow Trigger source has a single templated Instance that expands into 10 Instances, on each Flow Trigger execution all 10 Instances are sent through a pipeline. This is equivalent to manually creating all 10 Instances and assigning them to sources of the Flow Trigger.

Templated Instances can also be referenced as Condition sources. When referenced, the templated Instance expands and each source instance provides a unique condition instance by name. This is equivalent to manually creating all 10 Instances and assigning them to sources of the Condition.

When a templated Instance is referenced as an Instance expression, it expands into an array of all the objects. If this is not desired use parameters instead of templating and pass in the specific parameter (ex. `{{Connection.opc.tags({"tagid": "100"})}}`).

Templated Flow Triggers

Templated Flow Triggers are equivalent to creating many Flow Triggers with similar configuration. The Flow Trigger sources and trigger expression support template parameters.

A common use case for templated flow triggers is moving machine data on a state change of a machine. In this use case a flow trigger is configured with a trigger expression to trigger when the machine changes state. Once the trigger detects a state change, data for the machine is collected, modeled, and sent to the targets. Using trigger flow templates, the trigger expression and sources can be templated, passing a machineID parameter like `{{Connection.opc.machine({"machineID": {{this.machineID}}})}}`. The trigger flow template settings then define all the machines. This allows a single trigger flow configuration to flow data for 100's of machines.

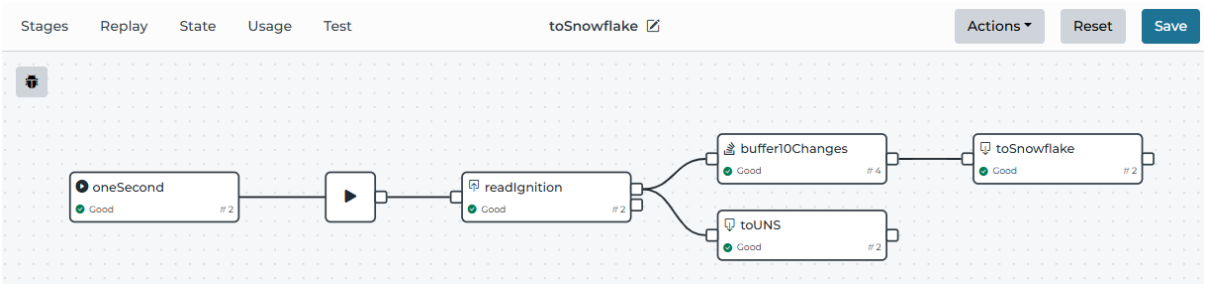
Pipeline | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/

Pipeline

Pipelines are the core engine for moving and transforming data. They define data flows that connect sources to targets, applying logic and transformations along the way. As an example, the pipeline below executes once a second, reads modeled data from Ignition, then sends it to a local UNS and to Snowflake, buffering the data to Snowflake to make writes more efficient.

Pipelines consist of Events, Triggers, and Stages, which are described below.



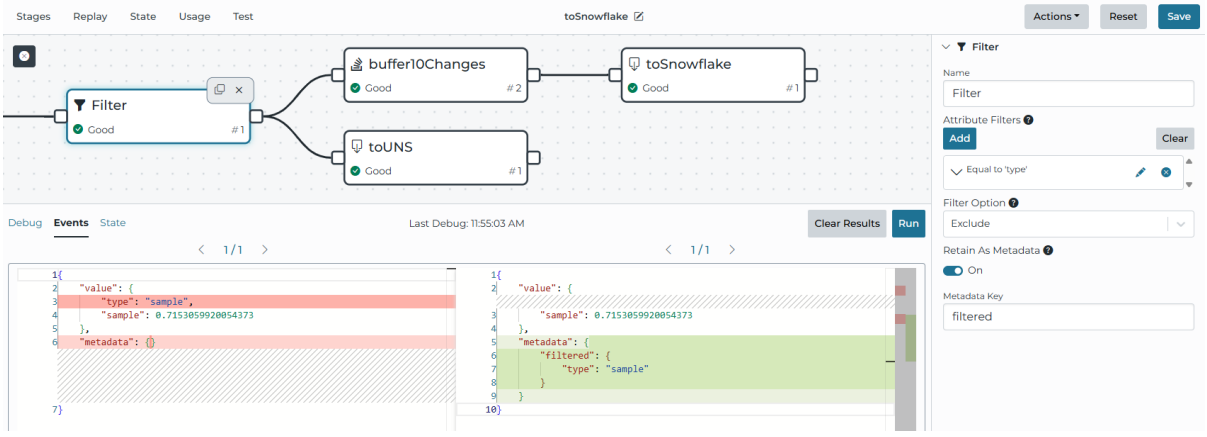
Events

When data enters a pipeline it's called an event. An event holds both a value and metadata. For example, if an event is from an OPC UA tag read, the value of the tag would be the event value and the tag source address would be metadata. Think of the event value as in what is sent over the wire when a pipeline write occurs. Event metadata is additional information in the event that can be used in the pipeline logic but is not part of the value.

For a more concrete example, consider the Filter stage below shown in Debug mode. Here we're sending an event through the Filter stage to remove the 'type' attribute from the value and place it in the metadata. On the bottom left you can see the event entering the filter stage, and the bottom right shows the event that exits the stage, with the difference shown by the coloring.

When this event reaches the toUNS or toSnowflake write stages, only the 'sample' attribute is included in the payload. The metadata could be used to control the MQTT topic, or other settings.

Events are generated by Triggers and processed sequentially through Stages.



Triggers

Triggers inject events into the pipeline. As examples, a Polled Trigger generates an empty event every time it fires, and an Event Trigger subscribes to an MQTT topic and generates an event any time data is published on the topic.

All pipeline events generated by triggers are sent to the start stage (play icon), which is the entry point of the pipeline. Note pipelines can have one or more triggers.

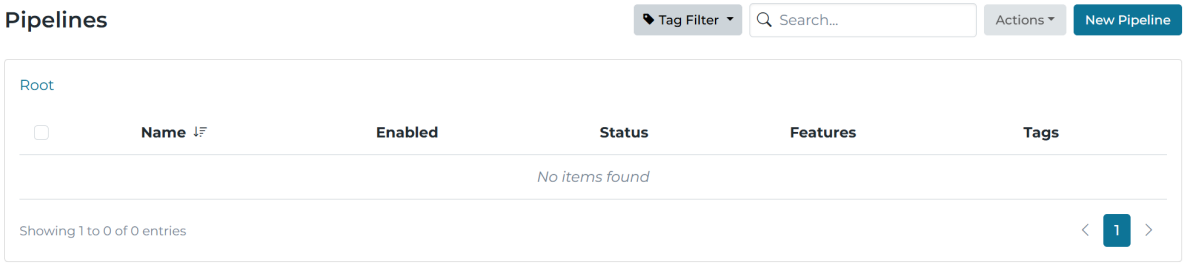
Stages

Stages perform I/O or transformations on events. For example, the Breakup stage takes an event value that's an Array and breaks it into one event for each row in the Array. The read stage uses the event value or metadata to issue a read (ex. machineID to read an OPC UA server) and outputs an event that's the result of the read.

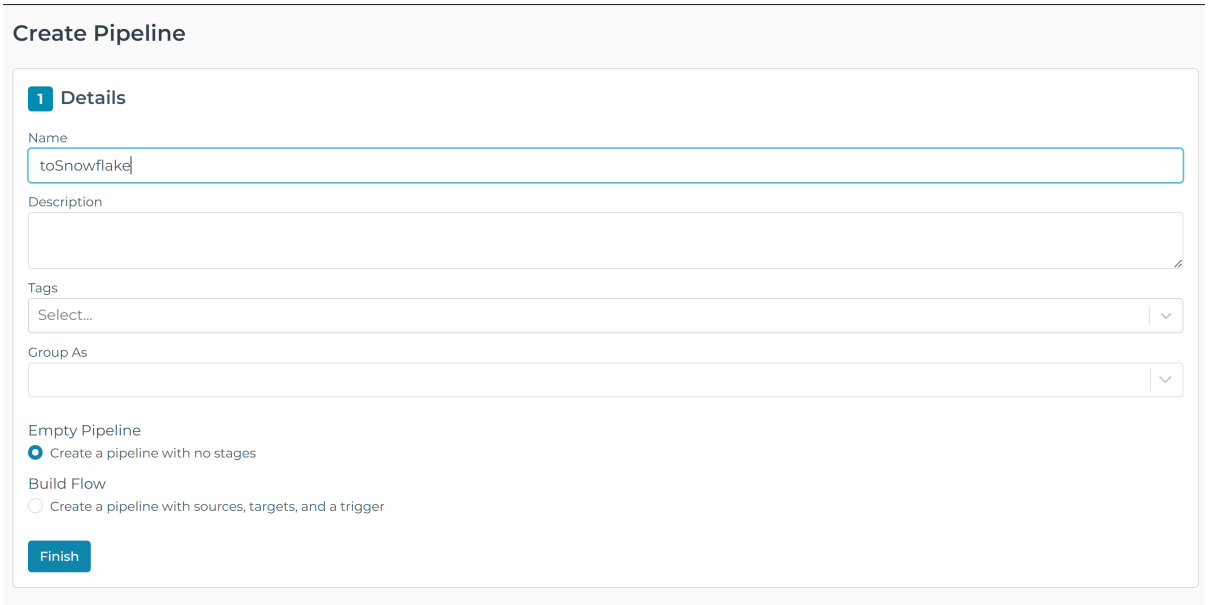
Stages take a single event in and can access either the event value or metadata. Stages can output 0..N events. When no events are output by a stage the pipeline execution stops for that event.

Creating a Pipeline

1. Click *Pipeline* in the configuration's Main Menu and then click *New Pipeline* to get started.



2. Enter a *Name* to represent the pipeline. Optionally enter a *Description*, *Tags*, and *Grouping*.
3. Choose to create an *Empty Pipeline* or choose *Build Flow*.
    - Empty Pipeline creates a pipeline with no triggers or stages that must be manually configured.
    - Build Flow continues the wizard and displays flow triggers settings to create a simple pipeline that reads and writes.
4. Once the Pipeline is created, navigate to *Stages* in the top left of the Pipeline to view and drag and drop Trigger or other Stages onto the Pipeline.



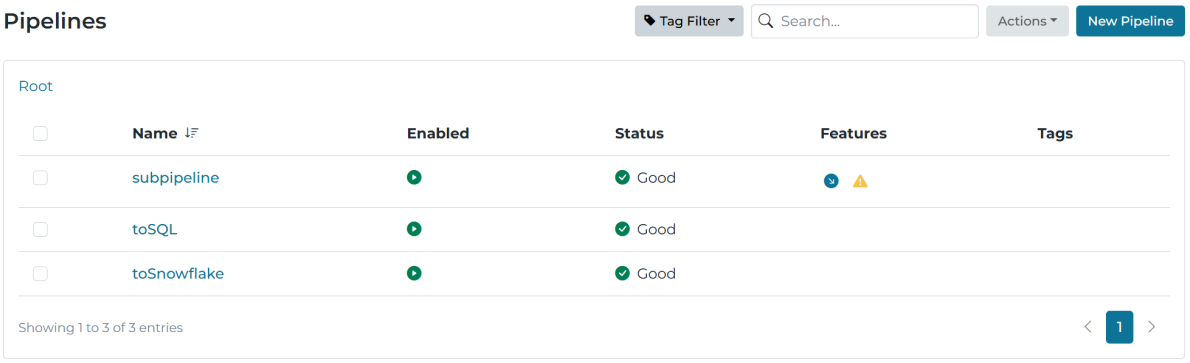
Building & Debugging Pipelines

Pipeline Debug mode allows you to simulate events flowing through the pipeline, and visually inspect what each stage does to the event. This can be very useful and intuitive way to understand and work with pipelines and stages.

For more information on how to use the feature, see the Debug section.

Pipeline Status

The list view shows the name, enabled status, processing status, active features, and tags for each pipeline. The name, enabled, and tags columns can be sorted by clicking on the column header. All triggers can be enabled/disabled on multiple pipelines at the same time by selecting them (using the checkboxes on the left) and then using an action from the Actions dropdown.



Column	Description
Name	The name of the pipeline or group
Enabled	This will show a play button if the pipeline has any enabled triggers, or if it is Callable.
Status	Status is based off of the last execution of the pipeline. **Not Started** - The pipeline has never run **Good** - The pipeline is executing successfully **Error** - An error occurred while processing **Warning** - The pipeline queue is growing faster than it is being processed.
Features	The feature icons indicate if a pipeline can be called internally, from the API, or if it has track changed enabled

Pipeline Stage Status

Each pipeline stage shows a status when viewing the pipeline, with the same status options as the overall pipeline status (i.e. good, not started, bad, etc). Each stage also shows the total execution count, which is the number events that have passed through the stage since it was last saved.

Pipeline Errors

When errors occur in a pipelines execution the stage where the error occurred and the overall pipeline go into error status. This can be seen on the pipeline list page and when opening the pipeline. Errors could include bad I/O (read or write), errors in transformation logic, etc.

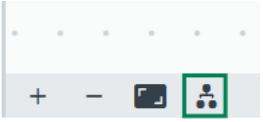
For more details on errors and overall error handling strategies for large deployments, see Pipeline Error Handling.

Helpful Features

Below is a collection of features that make working with pipelines easier.

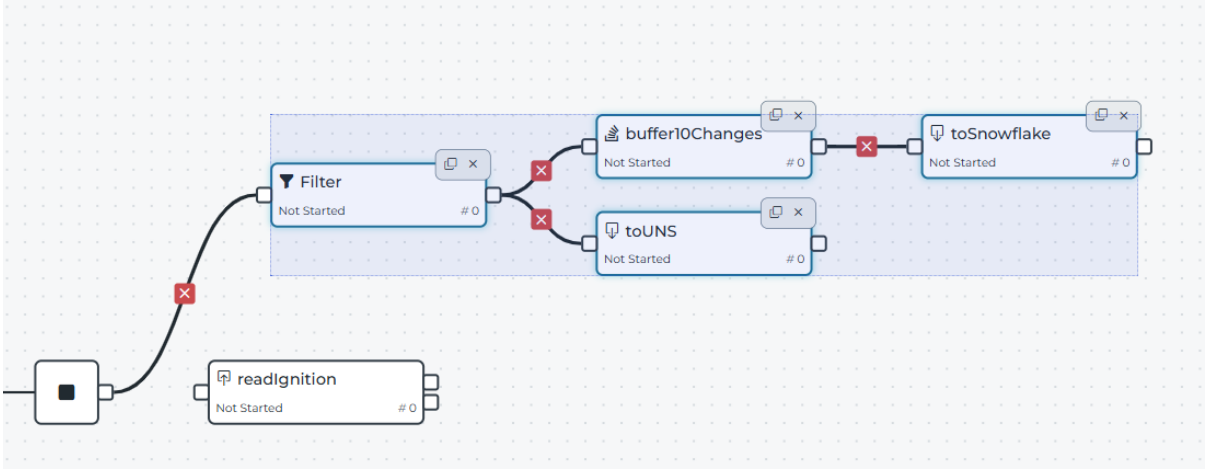
Auto Layout

For large pipelines, the auto-layout feature can help keep stages organized. You can find this in the lower left hand corner of the canvas.



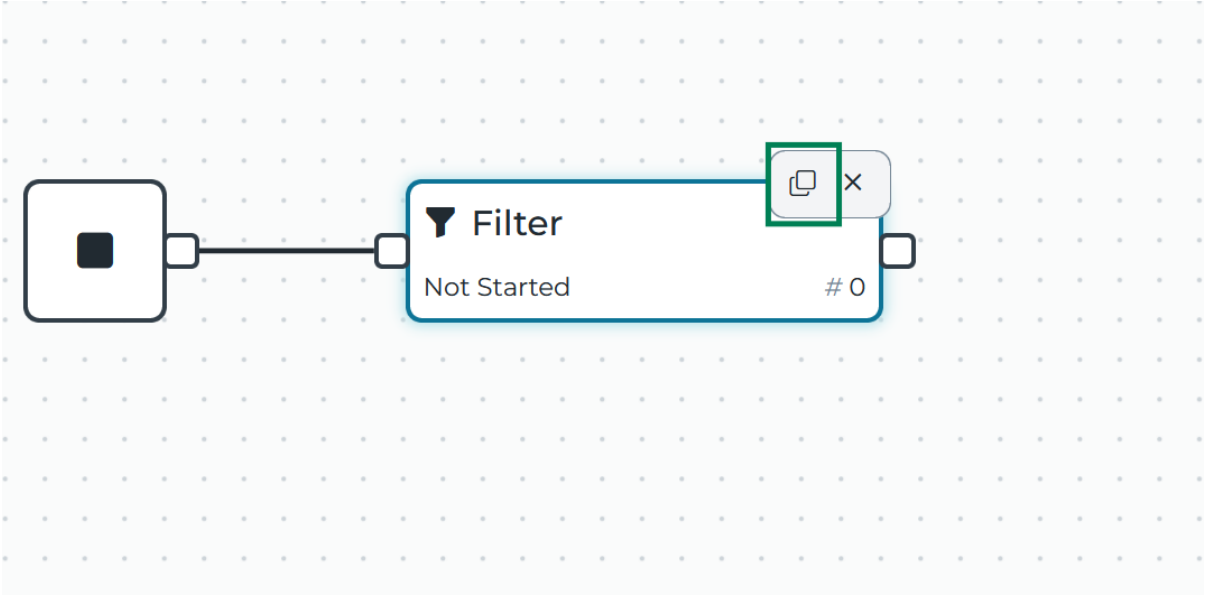
Multi-Select

The select multiple stages at once by either holding left-shift and dragging over the selected stages or holding left-control and selecting each stage.



Cloning Stages

To clone a stage, click on the desired stage and press the clone button.



Pipelines are a powerful tool to move and transform data. See the links below to learn more about pipeline components, debugging, monitoring, and more.

Stages | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/

Stages

- Transform
  - JSONata
  - JavaScript
- Common
  - Breakup
  - Filter
  - Flatten
  - Model
  - Model Validation
  - Size Buffer
  - Time Buffer
- Control
  - Delay
  - Error
  - For Each
  - Return
  - While
  - On Change
  - Switch
  - Subpipeline
- Parse
  - CSV
  - JSON
  - Parquet
  - XML
- Format
  - CSV
  - GZip
  - JSON
  - Parquet
  - Zip
- I/O
  - Read
  - Merge Read
  - Smart Query
  - Write
  - Write New

Common | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/common/

Common

Common Operations applied to event values that flow through a pipeline.

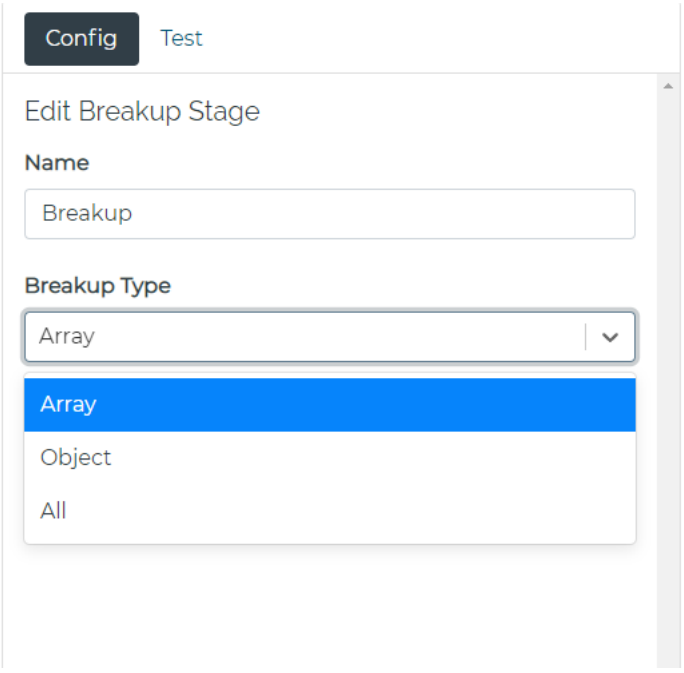
- Breakup
- Filter
- Flatten
- Model
- Model Validation
- Size Buffer
- Time Buffer

Breakup | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/breakup/

Breakup

The Breakup Stage is used to split an event value into multiple events.



Breakup Type

Break-up operation to perform on the event value. Array, Object, and All are supported.

Array

Split array value types into N events that contain a single scalar each. This operates on arrays of simple and complex types. Upon breakup, the stage sets `event.metadata.breakupIndex` to the array index of the value. An example Array Breakup operation is shown below. Continuously nested arrays can be broken up by setting a depth greater than one.

Depth

Defines how deep the break-up operation should traverse within the payload. This setting is particularly useful for deeply nested structures, such as multi-dimensional arrays or complex objects.

If the specified depth is deeper than the payload's structure, the operation will continue breaking it up until there's nothing left to break up.

Example Input Event Value

```
[{
    "a": 1,
    "b": 2,
    "c": 3
}, {
```

```
    "a": 10,
    "b": 20,
    "c": 30
}, {
    "a": 200,
    "b": 200,
    "c": 300
}]
```

Example Output Event Values

Value	Metadata
`{ "a": 1, "b": 2, "c": 3 }`	`{ "breakupIndex": 0, "breakupPath": "0" }`
`{ "a": 10, "b": 20, "c": 30 }`	`{ "breakupIndex": 1, "breakupPath": "1" }`
`{ "a": 200, "b": 200, "c": 300 }`	`{ "breakupIndex": 2, "breakupPath": "2" }`

Object

Split a complex value into N event values containing the values from each of its attributes. Upon breakup, the stage sets `event.metadata.breakupName` to the name of the attribute associated with the value. An example Object Breakup operation is shown below. Continuously nested objects can be broken up by setting a depth greater than one.

Example Input Event Value

```
{
 "a": 1,
 "b": 2,
 "c": 3
}
```

Example Output Event Values

Value	Metadata
1	{ "breakupName": "a", "breakupPath": "a" }
2	{ "breakupName": "b", "breakupPath": "b" }
3	{ "breakupName": "c", "breakupPath": "c" }

All

In this mode, if the event value is an array, it follows the array breakup example. If the event is an object, it follows the object example. When configured with a depth greater than one, the value will be recursively broken up.

Example Input Event Value

```
{
 "a": [1,2,3],
 "b": {
        "childOne": "valueOne",
        "childTwo": "valueTwo"
 },
 "c": [
        {
                "deepChild": "deepValue"
        }
 ]
}
```

Example Output Event Values

Value	Metadata
1	{ "breakupIndex": 0, "breakupPath": "a/0" }

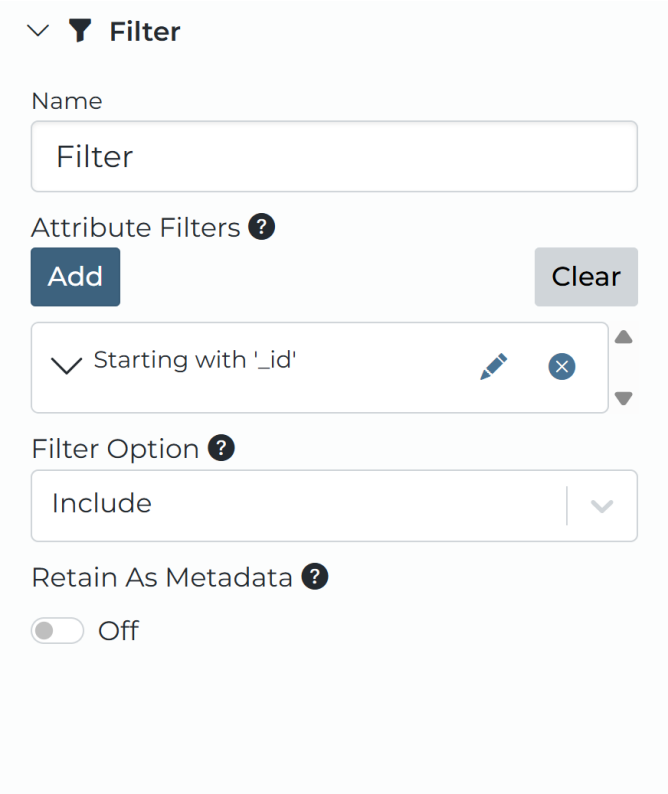
2	{ "breakupIndex": 1, "breakupPath": "a/1" }
3	{ "breakupIndex": 2, "breakupPath": "a/2" }
"valueOne"	{ "breakupName": "b/childOne", "breakupPath": "b/childOne" }
"valueTwo"	{ "breakupName": "b/childTwo", "breakupPath": "b/childTwo" }
"deepValue"	{ "breakupName": "deepChild", "breakupPath": "c/0/deepChild" }

Filter | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/filter/

Filter

The Filter stage filters data from the event. This makes it easy to move parts of the event from the value.



For example, consider data with a `SiteId` attribute. `SiteId` should be in the MQTT topic but not in the payload. A filter stage can move the `SiteId` field from the event value to metadata, and the output can reference metadata to include the `SiteId` in the MQTT topic.

The Filter stage works on complex data as well as Arrays of complex data. In the above example, if an array of objects with a `SiteId` attribute is passed to the filter, the `SiteId` attribute for each array object is filtered.

Filter List

Specify a list of filters to be applied to the payload in parallel. There are five types of filter: Exact Path, Starts With, Ends With, Contains, and REGEX.

Filter Option

The Filter stage can be set to either include or exclude keys that match the filter set. When set to include, keys are included in the payload if they match at least one of the specified filters. Likewise, when set to exclude, keys are excluded from the payload when they match at least one of the specified filters. The include and exclude option will apply to all specified filters within a filter stage, so for a combination of include and excluding, multiple filter stages in series are required.

Retain As Metadata

When enabled, anything removed from the event value is included in the event metadata under the key specified in **Metadata Key**.

In the above `SiteId` example, the MQTT topic output would be `{{event.metadata.filter.siteID}}`.

Metadata Key

Specifies the name of the key used to retain the filtered out data in the metadata object.

Filter Settings

Filter Text

Specify the text that the filter will use for comparison (e.g. setting the filter text to _id on a "Starts With" filter will match all keys that start with _id).

Recursive

Specifies whether the filter should be applied to only the root level of the event object, or to every level. When enabled, if a key is matched by a filter, all of its descendants will be included / excluded regardless of if they match the filter. Additionally, all keys of which the matched key is a descendant will also be included in order to maintain structure. Exact Path filters cannot be set to recursive.

Filter Types

Exact Path

Matches keys with the same full path in the format 'Parent.Child.GrandChild'. For example with an Exact Path filter of 'Parent.Child.GrandChild':

```
{
    "Parent": {
        "Child": {
            "GrandChild": "included",
            "GrandChild_Sibling": "excluded"
        },
        "Child_Sibling": "excluded"
    },
    "Parent_Sibling": "excluded"
}
```

Starts With

Matches keys that start with the provided filter text

Ends With

Matches keys that end with the provided filter text

Contains

Matches keys which contain the provided filter text

REGEX

Matches keys which satisfy a provided regular expression. For example, with the filter text "gr(a|e)y" the filter will match keys containing either "gray" or "grey". While this filter may be used for any regular expression, it is important to note that because

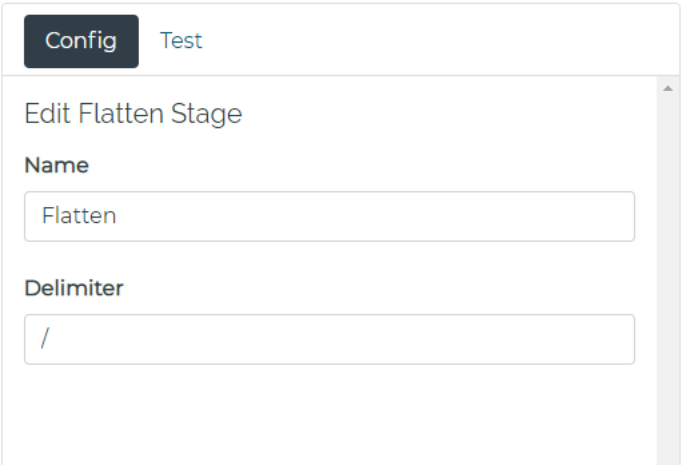
the filter may run against every key in the event, particularly complicated expressions or particularly large payloads may inversely affect the performance stage and slow the pipeline.

Flatten | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/flatten/

Flatten

The Flatten Stage can be used to compress a complex object hierarchy into an object that has attributes of simple types. During flattening the attribute path is used in the resulting attribute of the output object value. Flattening is applied recursively.



Delimiter

Delimiter to use between attributes when generating attribute names of the flattened object.

An example flatten operation is shown below

Flatten Input Event

```
{
 "machine": {
  "sensor1": {
   "v1": 1,
   "v2": 2.2
  },
  "sensor2": {
   "v1": 100,
   "v2": 20.2
  },
  "sensor3": {
   "v1": 1000,
   "v2": 2002.2
  }
 }
}
```

Flatten Output Event

```
{
```

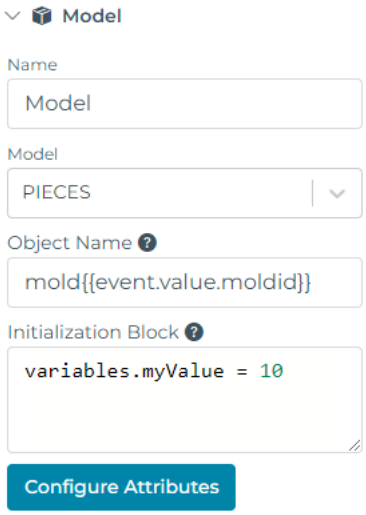
```
  "machine/sensor1/v1": 1,
  "machine/sensor1/v2": 2.2,
  "machine/sensor2/v1": 100,
  "machine/sensor2/v2": 20.2,
  "machine/sensor3/v1": 1000,
  "machine/sensor3/v2": 2002.2
}
```

Model | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/model/

Model

The Model stage is used to reshape data in an event data into a model definition without requiring an instance. Unlike instances, the Model stage can only work with data from the event. It cannot read data from other sources. If it's required to mix data from multiple sources, use a Read Stage followed by a Model Stage.



Model

The model used to shape the event data. Once set, select the `Configure Attributes` button to launch the model definition and map event data to each attribute.

Object Name

The name assigned to the event after it's been re-shaped. Note this is equivalent to an instance name, and populates the _name metadata.

Initialization Block

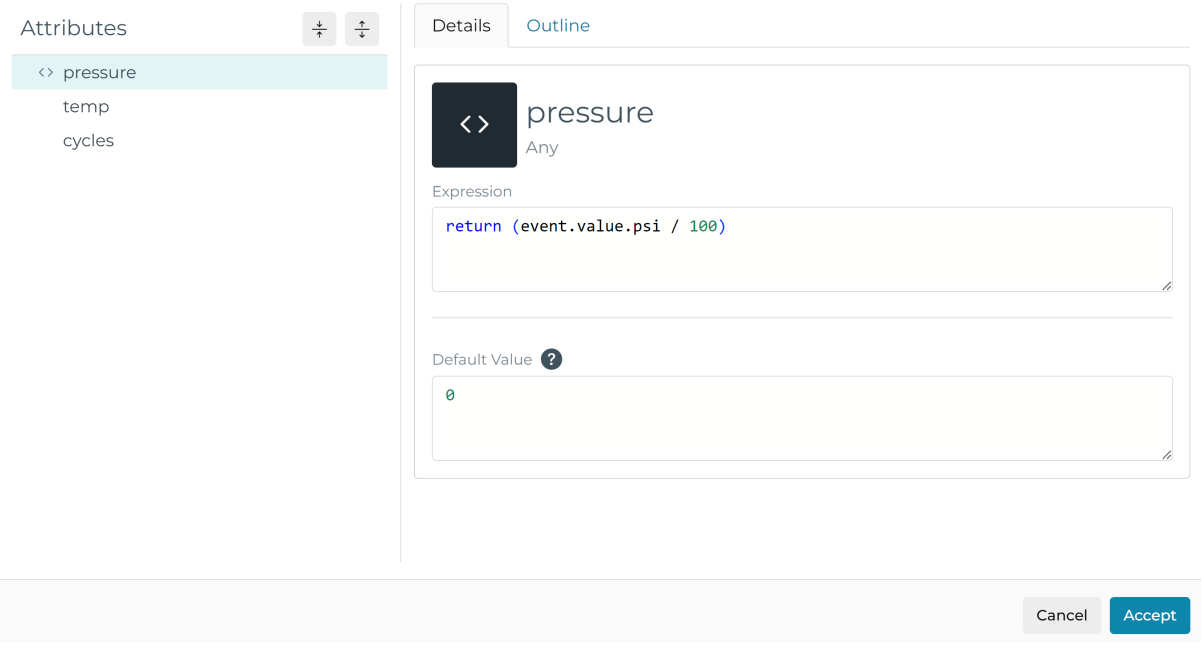
Similar to instances, the initialization block is used to set variables or calculated values that are shared across attribute expressions.

Example initialization block:

```
variables.mySetPoint = event.value.temp - event.value.tempOffset
```

Example attribute expression:

```
return (event.value.valveTemp - variables.mySetPoint)
```



Configuring Attributes

Select `Configure Attributes` to see all the attributes defined in the model and map each attribute to an event value. Both expressions and default values are available.

Expression

Similar to Instances, event data can be mapped to each attribute via a JavaScript expression. Note the {{}} syntax used in Instances is not required as the event is bound to a JavaScript object.

```
return (event.value.temp - event.metadata.pressureOffset)
```

Default Value

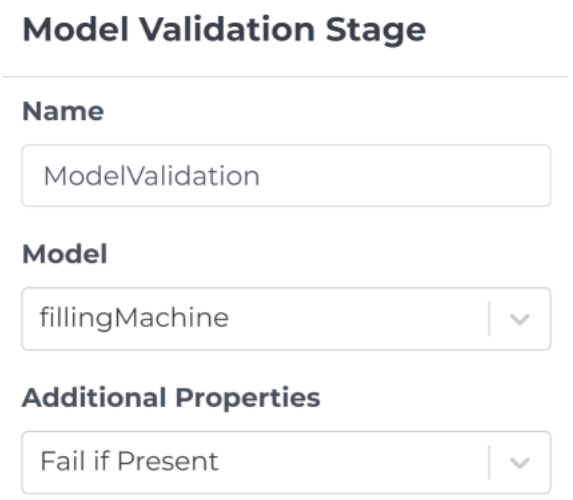
Use the default value to populate the value in cases where the expression is omitted or results in an error.

Model Validation | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/model_validtion/

Model Validation

The Model Validation can be used to validate an event value against a model definition and take a success or failure path. This is useful in cases where data has been modeled and needs to be validated against a model definition before sending it to the data lake or other consumer.



Model

The model to validate the event value against. The validation does no casting. For example, if an attribute value is String “123” and the model requires an integer, the validation fails. Attribute name comparison is case sensitive. Model default attribute values are injected into the payload if the attribute is missing.

Additional Properties

Controls if additional properties in the event value that are not in the model should fail or be allowed. For example, it may be important that the modeled properties be present in the event value (ex. site name) but additional data is allowed for the use case.

Examples

Assume the event value is based on an instance, and is shaped as followed.

```
{
  "pressure": 10.1,
  "speed": 90,
  "state": true,
  "locator": "line1"
}
```

The value is based off an instance of the Motor model, with the following definition.

```
{
        "name": "Motor",
        "attributes": [
            {
              "name": "pressure",
              "attributeType": "Internal",
              "internalType": "Real64"
            },
            {
                "name": "speed",
                "attributeType": "Internal",
                "internalType": "Int32"
            },
            {
                "name": "state",
                "attributeType": "Internal",
                "internalType": "Boolean"
            },
            {
                "name": "locator",
                "attributeType": "Internal",
                "internalType": "String",
                "required": true
            }
        ]
    }
```

The above payload passes the model definition. But the following variants fail.

Example, missing required field “locator”.

```
{
  "pressure": 10.1,
  "speed": 90,
  "state": true
}
```

Example, where speed is a float type and should be an integer.

```
{
  "pressure": 10.1,
  "speed": 90.999,
  "state": true,
  "locator": "line1"
}
```

Example, where an additional attribute is present and “Fail if Present” is set.

```
```

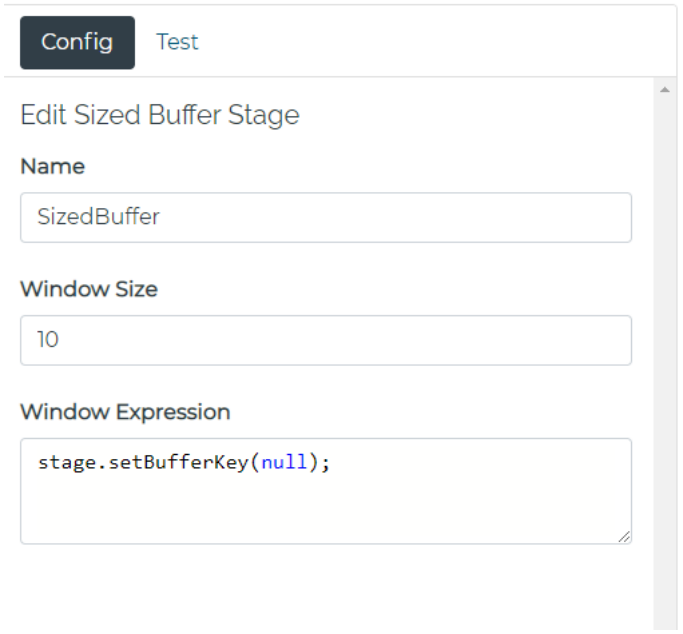
```
{
  "pressure": 10.1,
  "speed": 90,
  "state": true,
  "locator": "line1",
  "newAttribute": "this will fail validation"
}
```

Size Buffer | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/size_buffer/

Size Buffer

The Size Buffer stage is used to accumulate a max number of events and then emit all the events as an array. This is helpful in examples where you're sampling source data at a high frequency but the target system is more efficient when processing chunks at a time.



Window Size

Number of values to accumulate into the buffer before emitting the values to the next stage(s).

Timeout

In some cases you may want to accumulate N events or emit after Y period of time. Setting the Timeout will emit the buffer after Y period of time if it hasn't already hit the Window Size. Every time the buffer emits the Timeout is reset. When set to zero, there is no timeout.

Window Expression

The Window Expression allows for creating separate buffers given a unique key in the value or metadata. For example, you could create a buffer per Instance name or Model type.

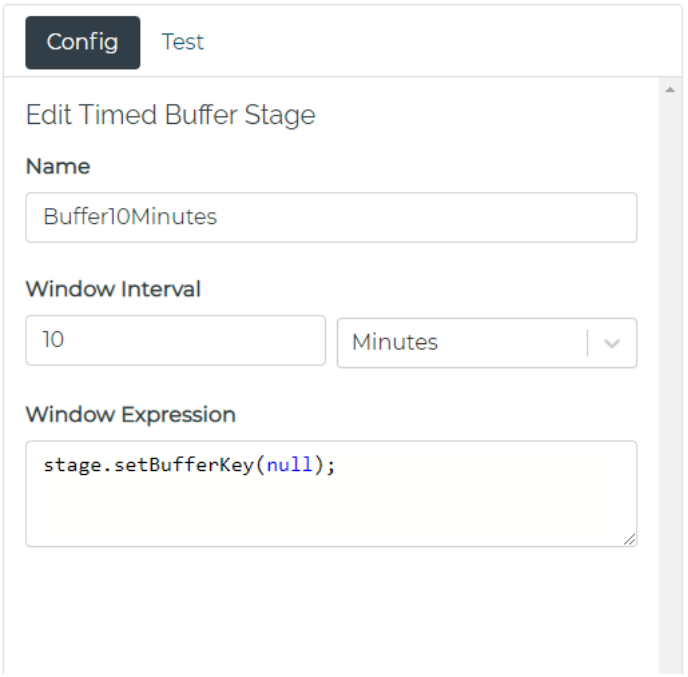
Setting the key is controlled using JavaScript and the `stage.setBufferKey` function. `stage.setBufferKey(null)` is the default, and creates a single buffer. `stage.setBufferKey(event.name)` creates a unique buffer per source name (i.e. input or instance name). Each unique buffer has its own Window Size and Timeout.

Time Buffer | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/common/time_buffer/

Time Buffer

The Time Buffer stage is used to accumulate events for a max period of Y, and then emit all the events as an array. This is helpful in examples where you're sampling source data at a high frequency, but the target system wants to process the data in time chunks (ex. 1 minute chunks).



Window Interval

Amount of time event data is accumulated before emitting.

Window Expression

See Size Buffer for details on the Window Expression.

Control | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/control/

Control

Stages to control the flow of events through the pipeline

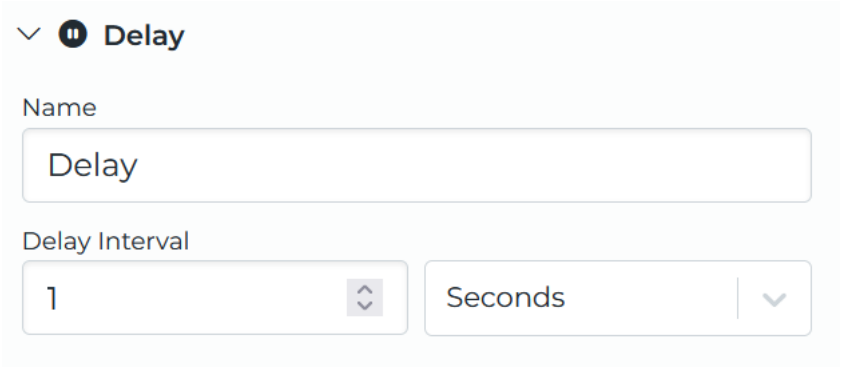
- Delay
- Error
- For Each
- Return
- While
- On Change
- Switch
- Subpipeline

Delay | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/delay/

Delay

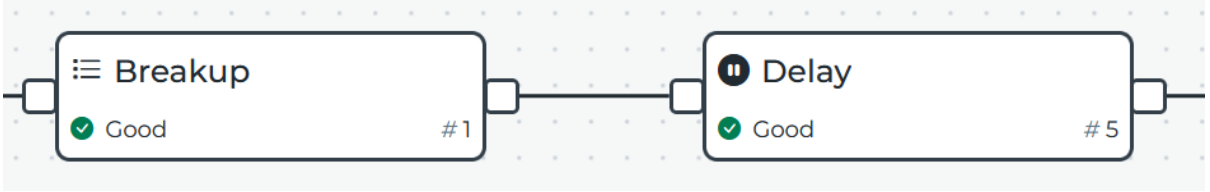
The Delay Stage is used to temporarily delay events



Non-blocking

The Delay Stage does not stop other event paths from moving forward.

For example, consider the following pipeline:



In this example the Breakup Stage will create 5 events. Then, each event will be sent to the Delay stage, which has a delay set to 10 seconds. The total execution time for these two stages will only be 10 seconds, not 50, since all of the events will be delayed in parallel.

Settings

Delay Interval

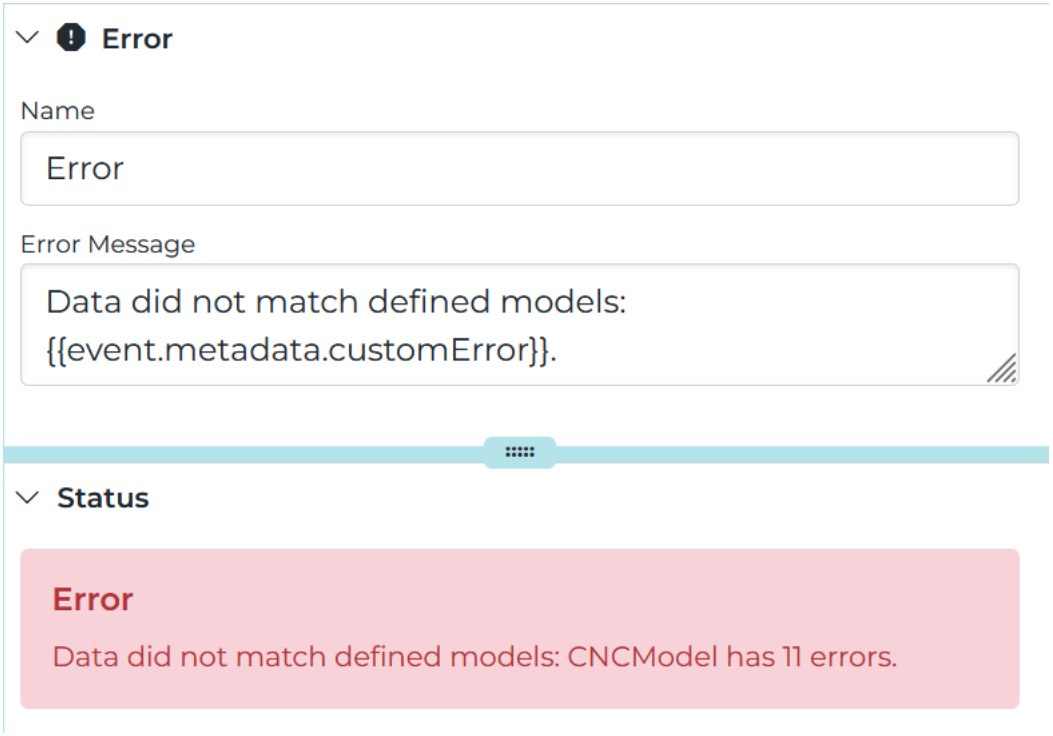
The amount of time to delay each event.

Error | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/error/

Error

The Error Stage is used to end pipeline execution with a custom error message



Error Message

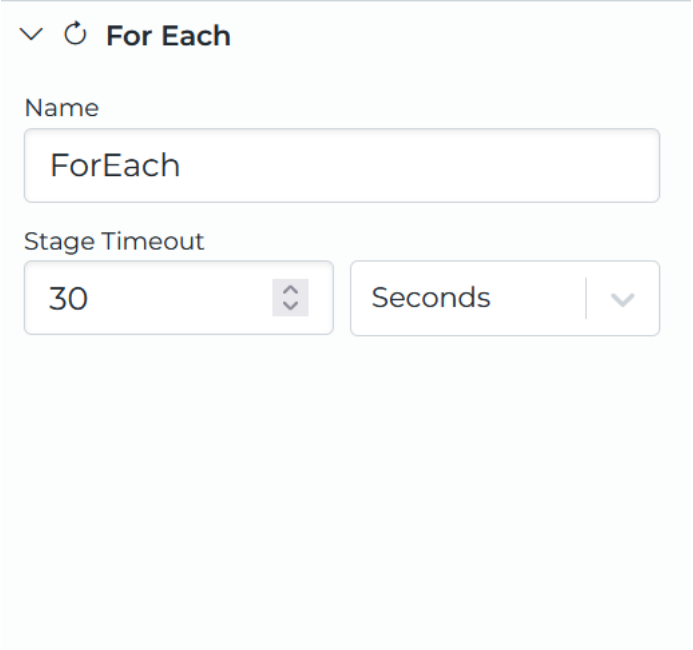
Specifies the error message output by the pipeline when ending execution.

For Each | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/for_each/

For Each

The For Each stage processes each element in an array, applies transformations to each value, and collects the transformed results while maintaining the original array order.



Stage Processing

This stage iterates over all values in the provided array.

Returning Values

Use the Return Stage to return a value during each loop iteration.

> Note: Each iteration of the For Loop must return a value.

Metadata Chaining

Metadata chaining ensures that updates to metadata are reflected in subsequent loop iterations. By returning modified metadata along with a value, the updated metadata object becomes available in the next iteration.

Metadata

During the loop, the metadata payload includes a `loopIndex` that indicates the index of the current iteration.

Success and Failure Paths

- **Success path:** If all iterations complete successfully, the generated output and updated metadata are passed down the success path.
- **Failure path:** If any iteration fails for reasons other than a timeout, the stage follows the failure path.

  Note: If a failure path is specified, all failures except a timeout will not cause the stage to go into an error state.

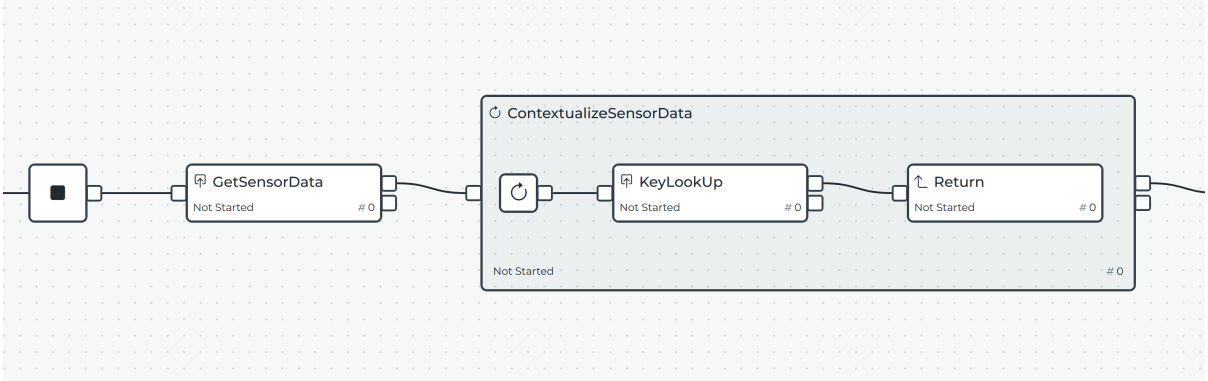
Settings

Stage Timeout

Defines the maximum time a stage can spend processing an event. If the timeout is reached before processing completes, all remaining operations are canceled.

Example

Use Case: Contextualizing Sensor Data with a Lookup Key



Description

Retrieve a set of sensor readings from a SQL table. Each reading includes a sensor ID and a raw value. The For Each stage is used to contextualize these readings by performing a lookup to retrieve additional metadata (ex. sensor location, unit of measurement) based on the sensor ID. The contextualized data is then passed to the next stage for further processing.

Incoming Value

```
[
  { "sensorId": "S1", "rawValue": 100 },
  { "sensorId": "S2", "rawValue": 200 },
  { "sensorId": "S3", "rawValue": 300 }
]
```

Outgoing Value

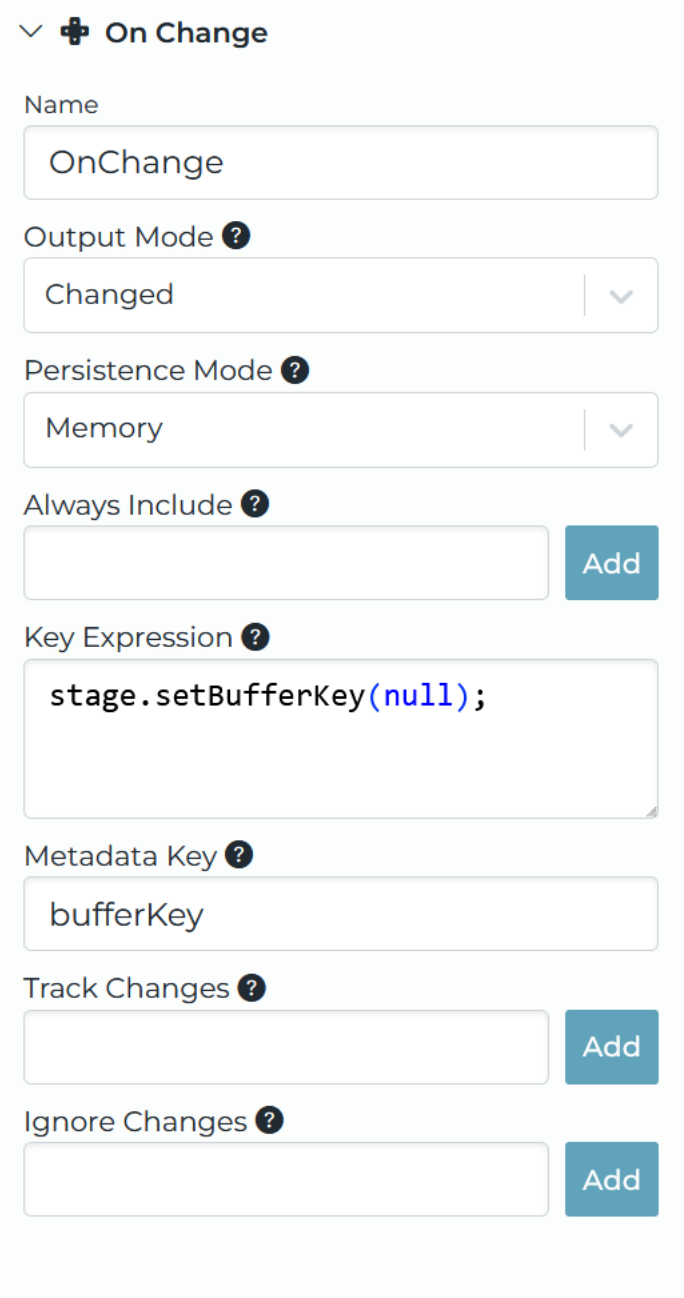
```
[
  { "sensorId": "S1", "rawValue": 100, "location": "Line 1", "unit": "Celsius" },
  { "sensorId": "S2", "rawValue": 200, "location": "Line 2", "unit": "Celsius" },
  { "sensorId": "S3", "rawValue": 300, "location": "Line 3", "unit": "Celsius" }
]
```

On Change | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/on_change/

On Change

The On Change stage is used to only send changed data.



The first event always passes through the stage. Subsequent events are compared against the last event's value. The cache is cleared any time the pipeline is saved or if the runtime is restarted.

Output Mode

All - Emit the entire event if anything in the event has changed. Set to *Changed* to only emit the changed values from the event. Values that remain the same are filtered out.
Changed - Filters only changed elements from complex payloads and arrays of complex payloads. If the payload is a simple array or not a complex payload, the entire payload is emitted.

Persistence Mode

Memory - The buffer key value is stored in memory and state will be lost if the pipeline or Intelligence Hub is restarted.
Disk - The buffer key value is stored in Pipeline State.

Always Include

When using the *Changed* **Output Mode**, this setting allows you to include parts of the event that may not have changed. For example, if the event contains an attribute `SiteId` and this never changes, add `SiteId` to the inclusion list to always include it in the emitted event.

Key Expression

The Key Expression allows you to create separate cache given a unique key in the value or metadata. For example, you could create a cache per Instance name or Model type.

Setting the key is controlled using JavaScript and the `stage.setBufferKey` function. `stage.setBufferKey(null)` is the default, and creates a single cache. `stage.setBufferKey(event.name)` creates a unique buffer per source name (i.e. input or instance name). New events are compared to the events unique cache.

Metadata Key

Specifies the name of the key used to retain the value passed to `stage.setBufferKey(value)` in the metadata object.

Note: When `null` is used as the value, then the value will not be retained in the metadata object.

Track Changes

Specifies the attributes to compare between two payloads. When combined with the *Changed* **Output Mode**, the changed payload will only be produced from the attributes that were compared. Attributes not included in the list will be ignored during the comparison.

Ignore Changes

Specifies the attributes to ignore when comparing between two payloads. If there is an overlapping attribute in **Track Changes** and **Ignore Changes**, the **Ignore Changes** list will take priority. For example, if **Track Changes** contains an attribute `data` and **Ignore Changes** contains an attribute `data.timestamp`, every attribute in `data` except `data.timestamp` will be compared.

Examples

Below is an example payload and how it works with *Changed*. Mode

First Payload: All of this data passes through the stage on the first event.

```
{
  "attr1": 123,
  "attr2": 123.123,
  "attr3": true,
```

```
  "attr4": "123",
  "attr5": [{"attr5_1": 1, "attr5_2": 2}, {"attr5_1": 1, "attr5_2": 2}],
  "attr6": {"attr6": 1, "attr7": 2}
}
```

Second Payload:

```
{
  "attr1": 1234,     // change here
  "attr2": 123.123,
  "attr3": true,
  "attr4": "123",
  "attr5": [{"attr5_1": 1, "attr5_2": 2}, {"attr5_1": 3, "attr5_2": 4}, {"attr5_1": 5, "attr
  "attr6": {"attr6": 1, "attr7": 3}   // change here
}
```

If the Output Mode is set to All, the full payload will pass through as is above because there are changes in it. If there were no changes, the event ends. If the Output Mode is Changed, the following event would pass through. Note only the changes pass through.

```
{
  "attr1": 1234,     // change here
  "attr5": [{"attr5_1": 1, "attr5_2": 2}, {"attr5_1": 3, "attr5_2": 4}, {"attr5_1": 5, "attr
  "attr6": {"attr7": 3}   // change here
}
```

For Complex Payload arrays, the comparison is done row by row. If a row changes, the entire row passes the filter. If the row doesn't change it is omitted. If there are new rows, these rows pass the filter. If rows are removed, but the remaining rows match the previous rows, nothing passes the filter.

First Payload: All of this data passes through the stage on the first event.

```
[
  {
    "attr_1": 1,
    "attr_2": 2
  },
  {
    "attr1_1": 3,
    "attr2_2": 4
  },
  {
    "attr1_1": 5,
    "attr2_2": 6
  }
]
```

Second Payload

```
[
  {
    "attr_1": 1,
    "attr_2": 2
```

```
  },
  {
    "attr1_1": 3,
    "attr2_2": 6
  },
  {
    "attr1_1": 7,
    "attr2_2": 8
  }
]
```

If the Output Mode is set to All, the full payload will pass through as is above because there are changes in it. If there were no changes, the event ends. If the Output Mode is Changed, the following event would pass through. Note only the changes pass through.

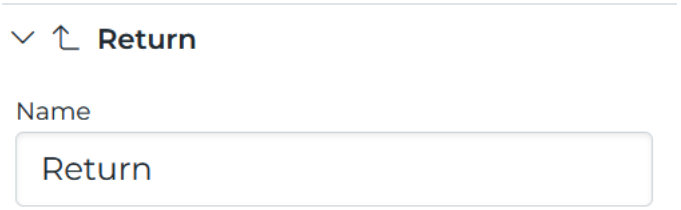
```
[
  {
    "attr1_1": 6
  },
  {
    "attr1_1": 7,
    "attr2_2": 8
  }
]
```

Return | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/return/

Return

The Return Stage sends data back to the source that called the pipeline, whether it's a Namespace Node, the REST Data Server, or another Pipeline. If multiple events are executing in parallel and writing to one or more return stages, the pipeline collects all returned values and sends them as a batch. If the pipeline execution ends in an error state, no values are returned, and an error is reported instead.

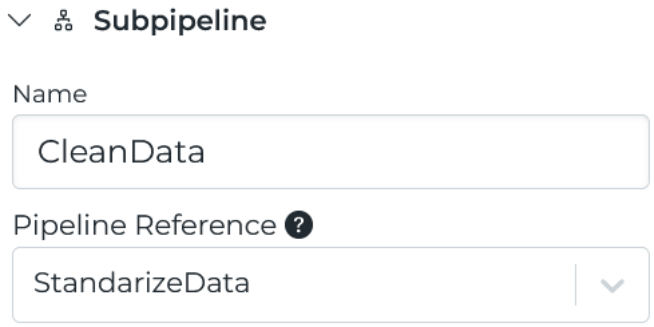


Subpipeline | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/subpipeline/

Subpipeline

The Subpipeline Stage passes the current value and metadata to a pipeline and returns the value(s) and metadata generated by the called pipeline. If the referenced pipeline encounters an error, the stage will either route the output through the failure path or stop pipeline execution in an error state if no failure path is defined. If the referenced pipeline returns one or more values, each is sent independently on the success path. If no values are returned, the pipeline stops executing in a successful state.



Settings

Pipeline Reference

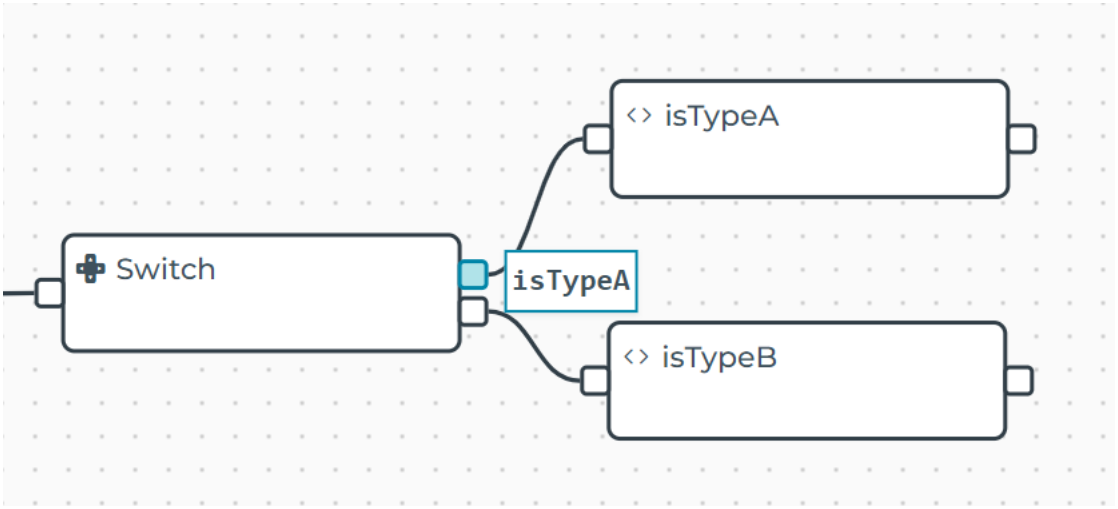
The name of the pipeline to call. Referenced pipelines must be “callable”.

Switch | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/switch/

Switch

The Switch Stage is used to conditionally send events to different stages.



Switch Cases

Labels

Optionally, each switch case can have a label. The label is used to identify the case in the UI.

Hovering over the output handle for a switch case will display that case's label.

Expressions

Each switch case has an expression. If the expression returns to `true`, the event is sent to all the stages connected to that case. If the expression returns `false`, then the next case will be evaluated.

The switch case expression is similar to a JavaScript Stage Expression, with a few notable differences:

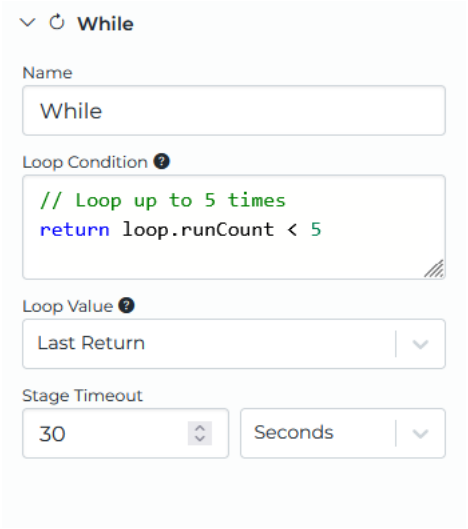
- The expression must return a boolean value.
- The expression cannot modify the payload.
- Only `event` and `system` can be referenced in the expression. (`stage` and `state` cannot be used)

While | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/control/while/

While

The While stage iterates over a sequence of transformations until a specified condition is met.



Stage Processing

This stage continues until the loop condition evaluates to `false` or an error occurs during processing.

Returning Values

Use the Return Stage to return a value during each loop iteration. Depending on the configured **Loop Value**, the returned value may be included in the generated output.

Metadata Chaining

Metadata chaining ensures that updates to metadata are reflected in subsequent loop iterations. By returning modified metadata along with a value, the updated metadata object becomes available in the next iteration.

Metadata

During the loop, the metadata payload includes a `loopIndex` that indicates the index of the current iteration.

Success and Failure Paths

- **Success path:** If all iterations complete successfully and the condition evaluates to `false`, the configured **Loop Value** and updated metadata are passed down the success path.
- **Failure path:** If any iteration fails for reasons other than a timeout, the stage follows the failure path.

  Note: If a failure path is specified, all failures except a timeout will not cause the stage to go into an error state.

Settings

Loop Condition

The loop condition is a JavaScript expression that determines whether the loop continues. Return `true` to continue processing or `false` to stop. The expression must return a boolean value; otherwise, the stage will produce an error. The condition is evaluated after each iteration.

```
/**
 * Loop Stage Variable
 *
 * Contains the result(s) of loop iterations.
 */
let loop: {
    /**
     * The number of times the loop has run
     */
    runCount: number,
    /**
     * The value associated with the most recent run
     *
     * Can be an array of values (Accumulated), the incoming value to the stage (Incoming),
     */
    value: any | any[],
    /**
     * The metadata associated with the latest run result
     */
    metadata: any
};
```

Loop Value

Controls the starting value for each loop and the output value of the stage. The available options are:

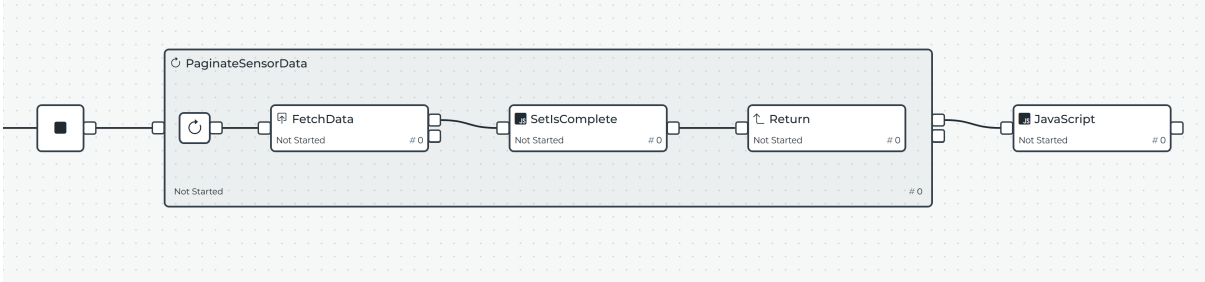
- **Accumulated:** All values returned from each iteration are collected in a single array (starting as `[]` on the first pass). If an array is returned during an iteration, it is flattened into the accumulated value, ensuring a one-dimensional array.
  - Common for pagination-style use cases when collecting array returns.
- **Incoming:** The original incoming value (e.g., `event.value`) of the While stage. This value remains unchanged by other values returned during looping.
  - Useful for scenarios where work is performed inside the loop, leveraging solutions like metadata chaining.
- **Last Return:** The output is the last value returned across all iterations. If no value is returned, the incoming value is used as the output.
  - Common for transactional use cases that wait for an update to happen

Stage Timeout

Defines the maximum time a stage can spend processing an event. If the timeout is reached before processing completes, all remaining operations are canceled.

Example

Use Case: Paginating Through Sensor Data



Description

Retrieve sensor data from a paginated API. Each page contains a subset of sensor readings, and the While stage is used to iterate through all pages until no more data is available. The loop condition checks the `isComplete` flag in the metadata to determine when the loop is done. The accumulated results are built up over multiple iterations.

Settings

- **Loop Condition:**

```
// Keep looping while isComplete has not been set
return !loop.metadata.isComplete
```

- **Loop Value:** `Accumulated`
- **Stage Timeout:** `30 seconds`

Iteration Details

Iteration 1

- **Starting Value:**

```
[]
```

- **Value Read:**

```
[
  { "sensorId": "S1", "value": 100 },
  { "sensorId": "S2", "value": 200 }
]
```

- **Metadata Returned:**

```
{ "loopIndex": 0, "isComplete": false }
```

- **Data Returned:**

```
[
  { "sensorId": "S1", "value": 100 },
  { "sensorId": "S2", "value": 200 }
]
```

Iteration 2

- **Starting Value:**

```
[
  { "sensorId": "S1", "value": 100 },
  { "sensorId": "S2", "value": 200 }
]
```

- **Value Read:**

```
[
  { "sensorId": "S3", "value": 300 },
  { "sensorId": "S4", "value": 400 }
]
```

- **Metadata Returned:**

```
{ "loopIndex": 1, "isComplete": false }
```

- **Data Returned:**

```
[
  { "sensorId": "S3", "value": 300 },
  { "sensorId": "S4", "value": 400 }
]
```

Iteration 3

- **Starting Value:**

```
[
  { "sensorId": "S1", "value": 100 },
  { "sensorId": "S2", "value": 200 },
  { "sensorId": "S3", "value": 300 },
  { "sensorId": "S4", "value": 400 }
]
```

- **Value Read:**

```
[
  { "sensorId": "S5", "value": 500 }
]
```

- **Metadata Returned:**

```
{ "loopIndex": 2, "isComplete": true }
```

- **Data Returned:**

```
[
  { "sensorId": "S5", "value": 500 }
]
```

Final Output Value

```
[
  { "sensorId": "S1", "value": 100 },
  { "sensorId": "S2", "value": 200 },
  { "sensorId": "S3", "value": 300 },
  { "sensorId": "S4", "value": 400 },
  { "sensorId": "S5", "value": 500 }
]
```

Format | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/

Format

Format event value into common file formats and/or compress files.

- CSV
- GZip
- JSON
- Parquet
- Zip

CSV | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/csv/

CSV

Converts the event into a CSV file. Once formatted, Output Pipeline stages can be used to write the file with a file-based connector such as *File* or *Amazon S3*.

Upon format the stage sets `event.metadata.fileFormat` to `csv`.

Note: Value with File datatype are represented with base64 encoded strings when using connectors that transmit data in `json` format.

Delimiter

The delimiter used when creating the file.

GZip | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/gzip/

GZip

The GZip stage compresses the event into a GZipped file value. This is useful for sending file data to cloud storage and can significantly reduce the size and time of transmission.

For example, common Pipeline patterns include buffering, formatting as CSV, and then GZip before sending to AWS S3 or Azure Blob storage.

JSON | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/json/

JSON

Converts the event into a JSON file. Once formatted, Output Pipeline stages can be used to write the file with a file-based connector such as *File* or *Amazon S3*.

Upon format the stage sets `event.metadata.fileFormat` to `json`.

Note: Value with File datatype are represented with base64 encoded strings when using connectors that transmit data in `json` format.

Parquet | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/parquet/

Parquet

Converts the event into a Parquet file. Once formatted, Output Pipeline stages can be used to write the file with a file-based connector such as *File* or *Amazon S3*.

Upon format the stage sets `event.metadata.fileFormat` to `parquet`.

Note: Value with File datatype are represented with base64 encoded strings when using connectors that transmit data in `json` format.

Zip | User Guide

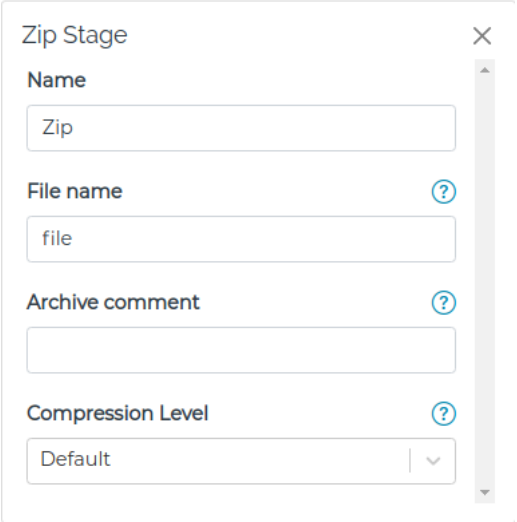
> Source: https://guide.highbyte.com/configuration/pipeline/stages/format/zip/

Zip

The Zip stage creates a ZIP archive containing a single file with the compressed event value. This is useful for sending file data to cloud storage and can significantly reduce the size and time of transmission.

For example, common Pipeline patterns include buffering, formatting as CSV, and then GZip before sending to AWS S3 or Azure Blob storage.

The name of the file included in the archive, archive comment, and compression level can be configured.



File name

Specifies the name of the file in the Zip archive. This field supports in-line references to allow for the file name to be based on the information from the input event. For example, assuming the input event has a metadata key called `filename` with value `process_data.csv`, setting the File name to `{{event.metadata.filename}}` would name the file in the archive `process_data.csv`. The default value is `file`.

Archive comment

Specifies the archive comment added to the Zip archive that is produced. This **optional** field supports in-line references.

Compression Level

Specifies the compression level to use. Higher compression may result in smaller archive sizes but take more time to generate. When Compression Level is set to **Default** a balance between size and speed is used.

I/O | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/io/

I/O

Read and Write data using existing connections.

- Read
- Merge Read
- Smart Query
- Write
- Write New

Merge Read | User Guide

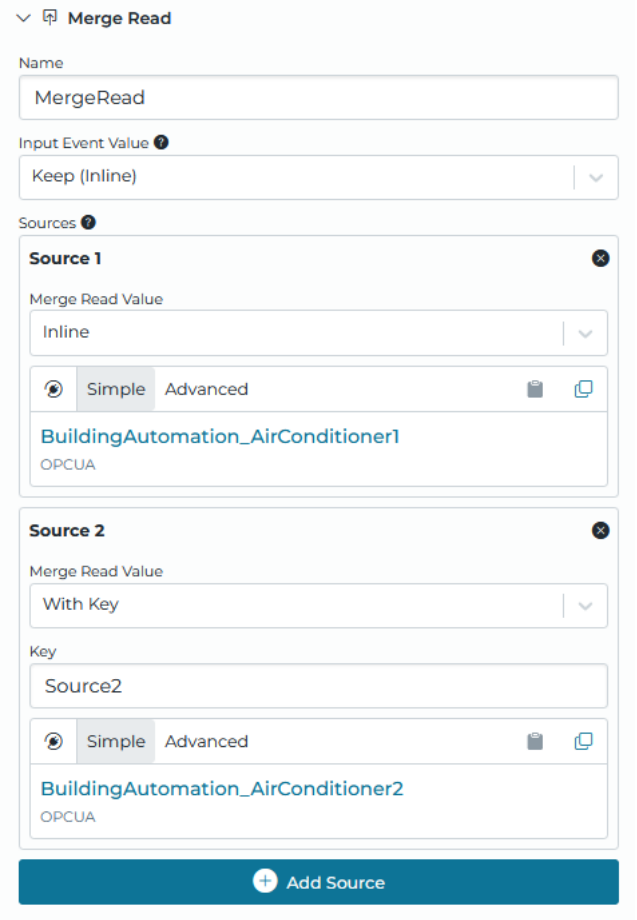
Source: https://guide.highbyte.com/configuration/pipeline/stages/io/merge_read/

Merge Read

The Merge Read Stage can be used to read Sources, combine the values into an object, and optionally include the input event value. The Merge Read Stage can pass parameters to Sources using static data or data from the input event.

This is useful for looking up data across sources. For example, an event may represent the state of a machine and include a batch identifier. The Read Stage can use the batchId to lookup batch information in the MES (ex. SQL) and then mix the result data with the raw event data.

The Merge Read Stage uses a list of Reference for it's sources, for more information on using References see: References



Input Event Value

Specifies whether or not the input `event.value` should be included with the output value.

Option	Description
Keep (Inline)	Keeps the input value attributes in the output event inlined with the data read from the sources. No hierarchy is added.
Keep (With Key)	Keeps the input value attributes in the output event under the attribute name provided by the Input Event Key. This adds one level of hierarchy to the output value.

Option	Description
Drop	Drops the input value from the output value.

See examples below for how these settings impact the output event.

Sources

One or more sources to read. Each source has the following configuration.

Setting	Description
Merge Read Value	Controls how the read value is placed in the output value. Options are Inline, or With Key.
Reference	A reference to the Source to read. For more information on passing parameters to references see: Reference Parameters

Each source is read in parallel, and the stage completes when all sources have been read.

Success / Failure

The Merge Read Stage provides alternative paths for successful and failed reads. This allows for pipelines to have special handling for read failures.

A failed read occurs if any source fails to read, and a single event is sent to the Failure path. The event includes the original event being read, as well as a `readResults` metadata value that contains the status of each read. The metadata looks as follows:

```
[{ // order matches the order of the target reference list
  reference: string   // the source reference name
  quality: string     // good or bad for success/failure
  reason: string      // the error message for the failed read
}]
```

If the Failure path is not connected, the stage produces an error. If all sources successfully read, the event takes the Success path.

Example

Assume an event enters the stage in the following shape.

```
{
  "machineId": 123,
  "temp": 108.2,
  "pressure": 90.2
}
```

Assume the stage is configured to read two other sources, with the Merged Read Value set as described below.

mes: Inline | SQL Input the parameter "machineid = {{event.value.machineId}} erp: With Key set to "erp" | An ERP Rest connection

Below are example output events given the Input Event Value setting.

```
```

```
// Keep (Inline)
{
  "machineId": 123,       // input value is inline
  "temp": 108.2,
  "pressure": 90.2,
  "operatorId": 555,      // mes read is inline
  "startTime": 8720,
  "erp": {                // erp is under 'erp' key
    "orderId": 8820
  }
}

// Keep (With Key). Assume the key is `machine`
{
  "machine": {            // input value is under 'machine' key
    "machineId": 123,
    "temp": 108.2,
    "pressure": 90.2
  },
  "operatorId": 555,      // mes data is inline
  "startTime": 8720,
  "erp": {                // erp is under 'erp' key
    "orderId": 8820
  }
}

// Drop
{
  // event value is omitted
  "operatorId": 555,      // mes data is inline
  "startTime": 8720,
  "erp": {                // erp is under 'erp' key
    "orderId": 8820
  }
}
```

Merge Read Stage Errors

If all sources are read and there is an attribute name conflict in the resulting output value, the stage produces an error. For example, if the input event has an attribute named myAttribute and is set to Inline, and a source reads data back that has an attribute named myAttribute and is also set to Inline, this produces a stage error. To resolve this, use a unique key for the input value or source.

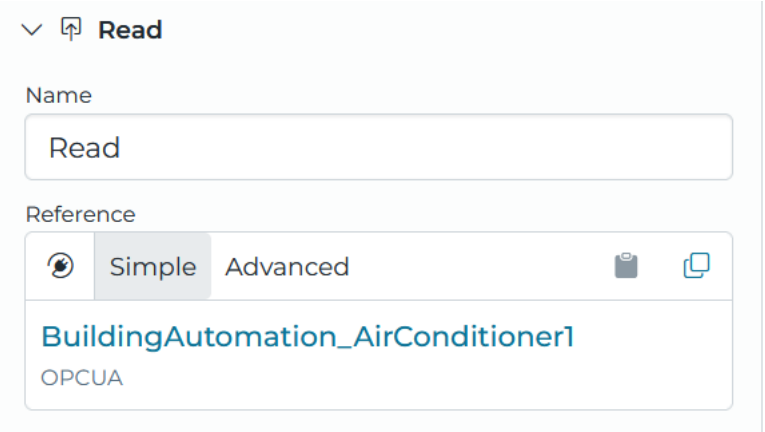
Read | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/io/read/

Read

The Read Stage can be used to easily read a single source. For more complex read operations, use the Merge Read Stage.

The Read Stage uses a Reference as it's source, for more information on using References see: References



Reference Parameters

If a reference that supports parameters is used in this stage the parameters can be dynamically driven using the incoming event with the following syntax: `{{event.value}}`

This syntax can be combined with static values, for instance `index_{{event.metadata.breakupIndex}}` if the breakup index was "10", this would result in "index_10".

For more information on using the `event` variable see: Using Transform Expressions (Note that `event` is the only variable is accessible from this stage)

Success / Failure

The Read Stage provides alternative paths for successful and failed reads. This allows for pipelines to have special handling for read failures.

A failed read occurs if the source fails to read, and a single event is sent to the Failure path. The event includes the original event being read, as well as a `readResults` metadata value that contains the status of the read. The metadata looks as follows:

```
{
    reference: string    // the source reference name
    quality: string      // good or bad for success/failure
    reason: string       // the error message for the failed read
}
```

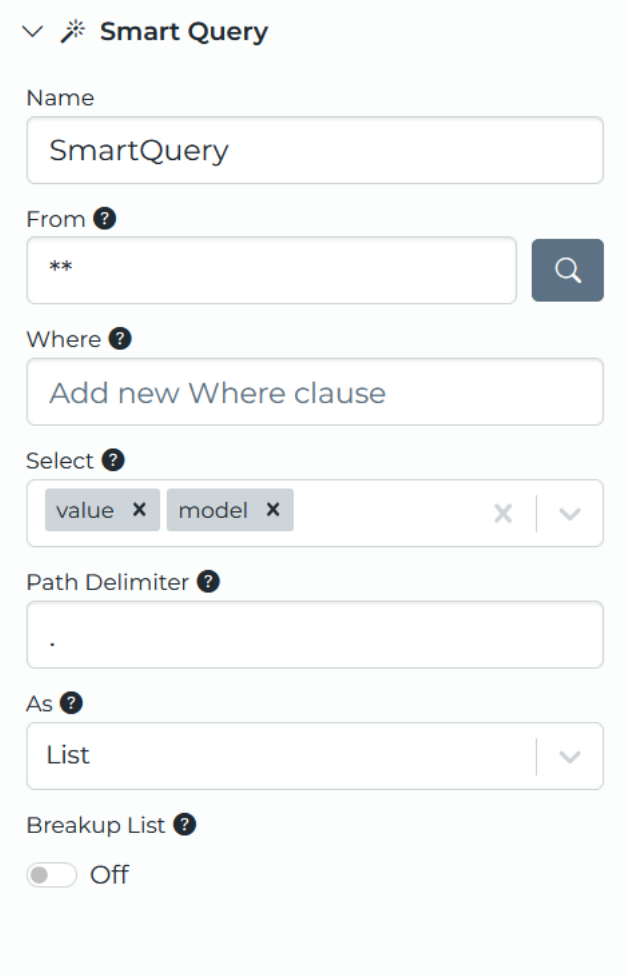
If the Failure path is not connected, the stage produces an error. If the source is successfully read, the event takes the Success path.

Smart Query | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/io/smart_query/

Smart Query

The Smart Query Stage is used to read the Namespace. The stage queries all or part of a namespace, filters, and transforms the results to make pipeline processing easy. The stage uses JSONata syntax to query the namespace.



From

Controls what part of the namespace is queried. This is similar to a SQL FROM clause, but using JSONata syntax. Below are some examples.

From Query	Description
Boston.Paint.Line1.*	This is the configuration Namespace, and gets Line1 and all it's children.
Boston.Paint.Line1.**	Gets Line1 and all of its children, recursively.
HighByte.Instance.*	This is the default namespace, and represents the HighByte configuration. This queries all Instances in the configuration.
**	Get the root of the namespace and all child nodes, recursively.

To populate the **From** field using a browsing experience, click on the magnifying glass icon located to the right of the field.

Where

Applies conditional logic to the From clause to filter the query. This is similar to a SQL WHERE clause. For example, you can filter for all instances modeled by Conveyor, or all Instances that have an attribute named `state` set to true.

Where	Description
*[model="Conveyor"]	Select all instances modeled after Conveyor.
*[value.state=true]	Select all instances with an attribute named `state` set to true.
*[value.state=true and model="Conveyor"]	Logically combine both queries using and/or.

Select

Controls what data is returned by the query. This is similar to a SQL SELECT clause, except it cannot be used to return specific attributes of an Instance, only the full value. The available options are described below.

Select	Description
value	Returns values in the namespace, similar to the test read results.
path	Returns the paths in the namespace. For example, [Boston.Paint, Boston.Paint.Line1, ..].
name	Returns the name of the source. For a modeled source this would be the Instance name.
model	Returns the name of the model for each node, and an empty string if the data isn't modeled.

Path Delimiter

Controls the Delimiter used when building the path from the namespace. By default this is a "." (ex. Boston.Paint).

As

Controls the shape of the resulting read. For some workflows, hierarchy is required, and for others an array of values with a path attribute is preferred.

Note the difference between "Inline" and non-inline is the inclusion of a "value" node for attributes. See below examples.

As Option	Description
Hierarchy	The reads return the full hierarchy of the namespace.
List	Returns an array of all the nodes.

Smart Query Metadata

The event generated by the stage includes metadata for path, name, and model under `event.metadata.query`. This is useful in cases where the metadata isn't needed in the payload, but it is needed in the event to control the topic path the write occurs on, buffering, or other data flow needs.

Smart Query Examples

For the below examples, assume the Namespace looks as follows. Conveyor is based on the Conveyor model, and Tank on the Tank model.

```
{
```

```
    "Boston": {
        "Paint": {
            "Line1": {
                "Conveyor": {
                    "speed": 100,
                    "state": 50
                },
                "Tank": {
                    "level": 50
                }
            }
        }
    }
}
```


Example *As* results

For these examples, Select is set to value only.

As: Hierarchy


```
{
  "Boston": {
    "Paint": {
      "Line1": {
        "Conveyor": {
                    "speed": 100,
                    "state": 50
        },
        "Tank": {
                    "level": 50
        }
      }
    }
  }
}
```


As: List


```
[
  {
    "speed": 100,
    "state": 50
  },
  {
    "level": 50
  }
]
```


Select all instances modeled after conveyor

Select: Boston.Paint.Line1.* Where: *[model = "Conveyor"] Select: value, path, model As: List


```
[
```

```
  {
        "speed": 100,
    "state": 50,
    "_path": "Boston.Paint.Line1.Conveyor",
    "_model": "Conveyor"
  }
]
```


Select all instances that have an active attribute set to true

Select: Boston.Paint.Line1.* Where: *[value.speed = 100] Select: value As: List

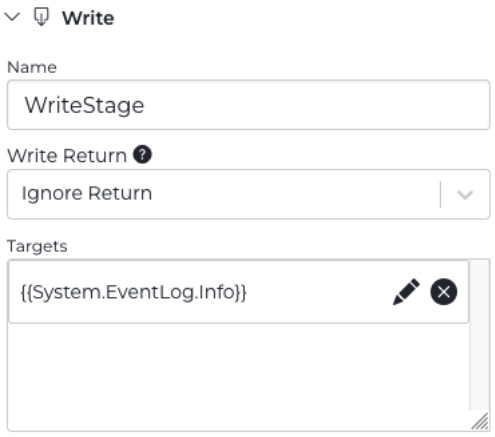

```
[
  {
    "speed": 100,
    "state": 50
  }
]
```

Write | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/io/write/

Write

The Write Stage writes events to an existing Output.



Targets

The list if outputs to write to. There must be at least one. Use the reference panel to drag & drop outputs.

Write Return

In cases where you don't care about handling the success/failure of a write, set this to Ignore Result. In this mode the pipeline hands the write to the connection thread and continues processing. If there is a write failure, the Pipeline status will update with the failure.

If you need to control success/failure paths, set this to Wait for Completion. The pipeline hands the write to the connection thread, but then waits for the I/O to complete. See below on logic for Success/Failure paths.

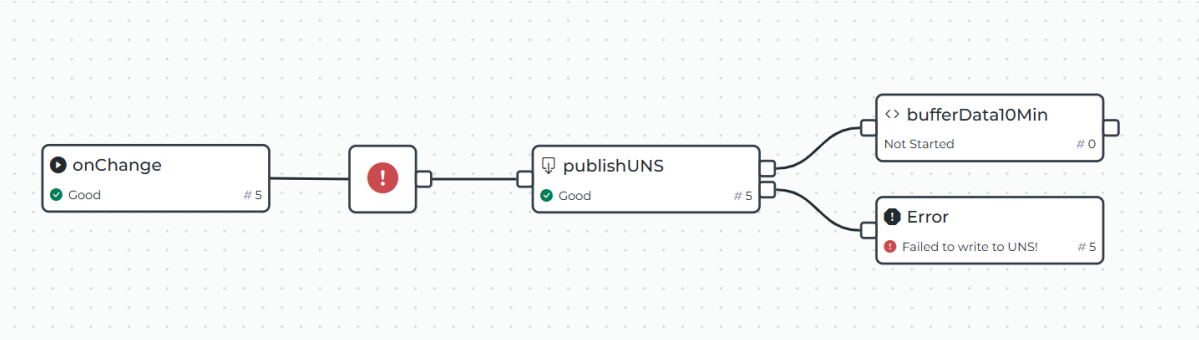
Wait for Complete will cause the pipeline to block waiting for the I/O to finish. This can significantly slow down pipeline performance. Only use this setting if you need to perform custom logic in the event of failures.

Success / Failure

When Wait for Complete is set, the stage provides alternative paths for successful and failed writes, allowing the pipeline to have special handling for write failures.

If all targets are written to successfully, the pipeline takes the success path. If a single target fails to write, the pipeline takes the failure path.

If the failure path is not connected, the write stage produces an error. If it is connected, a single event is sent to the failure path. Below is an example of a failure being sent to an error stage.



If Store & Forward is enabled for the connection and the write fails, the pipeline takes the failure path. The write is still inserted into the store & forward queue and is sent when the connection returns to good status.

The failure event includes the original event being written, as well as a `writeResults` metadata value that contains the status of each write. The metadata looks as follows:

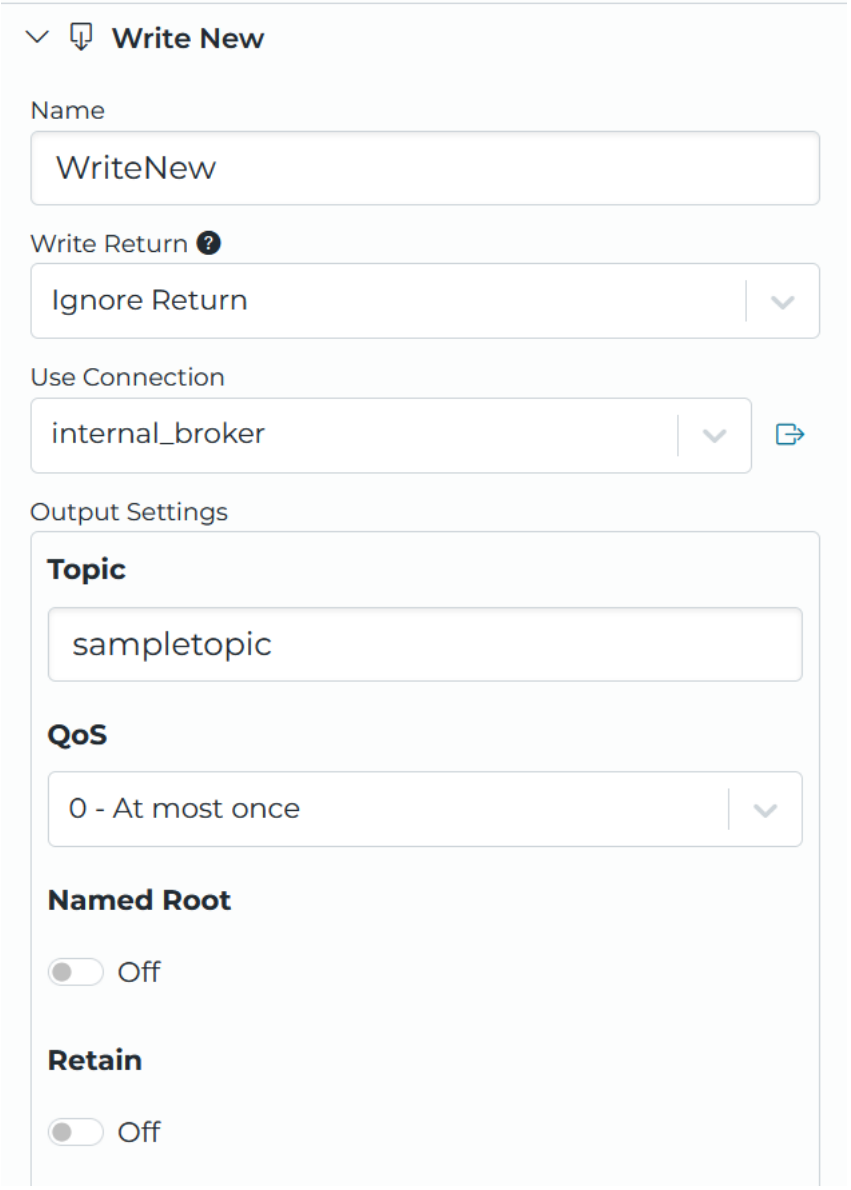
```
// order matches the order of the target reference list
[
  {
    reference: string,    // the target reference name
    quality: string,      // good or bad for success/failure
    reason: string        // the error message for the failed write
  }
]
```

Write New | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/io/write_new/

Write New

The Write New stage writes events to an Output that doesn't exist under the Connection. This is a more flexible away to control the Output settings and is recommended if Outputs definitions are not being shared between Pipelines.



Write Return

See the Write Stage for Write Return and success/failure paths.

Connection

Use the reference panel to define the Connection to use for the write. This controls the available output options.

Write Properties

Once a Connection is selected define connection-specific settings.

The properties of the new Target can be defined in three ways:

1. Static/constant settings
2. Dynamically using `{{}}` expressions like `{{event.value.topic}}`. When defining properties this way, content within the `{{}}` is evaluated as a Javascript. The expression can use `event` and `System` object. As a more advanced example, an MQTT topic setting could be defined as `{{event.metadata.baseTopic}}/{{event.value.name}}/{{new Date().toISOString()}}`
3. Dynamically using the Write Properties Expression field. This is only required for advanced use cases that require additional Javascript to generate the property settings. See below for additional details.

Advanced: Using a Write Properties Expression

Advanced use cases may require additional Javascript in order to generate the properties required to define the new Target. In the *Write Properties Expression*, use the `event` and `stage` objects to set properties based on `event.value`. This feature is most useful when setting properties that are not simple values, like arrays for example.

Examples below show how to set write values for single properties or multiple properties at once.

```
// Set a single topic for MQTT
stage.setWriteProperty("topic", "my/topic");

// Set multiple values
stage.setWriteProperties({topic: "my/topic", qos: 0});
```

Parse | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/parse/

Parse

Parse stages convert string formats (ex. JSON/XML/CSV Strings) or file-like formats (ex. Parquet Files or Base64 encoded strings of CSV files) into JSON values.

- CSV
- JSON
- Parquet
- XML

CSV | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/parse/csv/

CSV

Converts an `event.value` that is either an encoded CSV file or an unencoded CSV string, into JSON.

After passing through this stage `event.value` will hold the parsed CSV. Properties can then be accessed using `event.value.myProperty`.

For more information on using

Incoming Types

File

Processes events that are encoded files. If *File Encoding* is set to `Auto` the file's BOM will be inspected to determine the encoding type, if the file is missing a BOM this may have to be set explicitly.

String

Processes events that are unencoded stringified CSV.

Settings

Delimiter

Specifies the delimiter used to separate headers and fields. The default delimiter is a comma. Only single character delimiters are supported.

Quote

The quote character used by the CSV parser. Defaults to ". Supports the following escaped characters \t (tab), \r (return), \n (newline). Leave this blank to disable quoting.

Escape

The escape character used by the CSV parser. Defaults to \. Supports the following escaped characters \t (tab), \r (return), \n (newline). Leave this blank to disable escaping.

Override Headers

List of headers used to replace the file headers, or provide headers (e.g., header1, header2, header3).

Note: There must be a header for each column of data.

Custom Error Handling

A stage can be attached to the "failure" path to handle parse failures. In the event that the parse fails the original event will continue down the failure path unchanged, and a new metadata attribute `event.metadata.failures` will be added with an array of messages describing why the parse failed.

JSON | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/parse/json/

JSON

Converts an `event.value` that is either an encoded JSON file or a JSON string, into JSON.

After passing through this stage `event.value` will hold the parsed JSON. If the JSON was an object, for example, properties could be accessed using `event.value.myProperty` .

Incoming Types

File

Processes events that are encoded files. If *File Encoding* is set to `Auto` the file's BOM will be inspected to determine the encoding type, if the file is missing a BOM this may have to be set explicitly.

String

Processes events that are unencoded stringified JSON.

Custom Error Handling

A stage can be attached to the "failure" path to handle parse failures. In the event that the parse fails the original event will continue down the failure path unchanged, and a new metadata attribute `event.metadata.failures` will be added with an array of messages describing why the parse failed.

Parquet | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/parse/parquet/

Parquet

Converts an `event.value` that is a Parquet string, into JSON.

After passing through this stage `event.value` will hold the parsed Parquet file. Properties can be accessed using `event.value.myProperty` .

Custom Error Handling

A stage can be attached to the "failure" path to handle parse failures. In the event that the parse fails the original event will continue down the failure path unchanged, and a new metadata attribute `event.metadata.failures` will be added with an array of messages describing why the parse failed.

XML | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/parse/xml/

XML

Converts an `event.value` that is either an encoded XML file or a XML string, into JSON.

After passing through this stage `event.value` will hold the parsed XML. Properties can be accessed using `event.value.myProperty` .

Incoming Types

File

Processes events that are encoded files. If *File Encoding* is set to `Auto` the file's BOM will be inspected to determine the encoding type, if the file is missing a BOM this may have to be set explicitly.

String

Processes events that are unencoded stringified XML.

Custom Error Handling

A stage can be attached to the "failure" path to handle parse failures. In the event that the parse fails the original event will continue down the failure path unchanged, and a new metadata attribute `event.metadata.failures` will be added with an array of messages describing why the parse failed.

Transform | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/stages/transform/

Transform

Transform stages let you take an incoming value, process or modify it in some way, and then pass the updated result to other stages in the pipeline.

- JSONata
- JavaScript

Like a Custom Condition, transform stages manipulate data based on a provided expression. However, Transforms only operate on the event data. Use Transforms to shape or enrich events in a pipeline, whether the event contains raw input data or modeled data. Use Custom Conditions to clean up raw input before modeling or sending it to a pipeline.

Using Transform Expressions

Both the JavaScript and JSONata transform expressions have access to the following objects:

Event

The `event` variable contains read-only information about input event. Its type is described below.

```
class Event {
    /**
     * Unique ID of the parent event in form XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
    **/
    parent: string;

    /**
     * Unique ID of the current event in form XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
    **/
    id: string;

    /**
     * The event value. This is the value written to the pipeline for the first stage and va
    **/
    value: any;

    /**
     * Alias set by the Source when the value was input.
    **/
    alias: string;

    /**
     * Describes the quality of the value (typically Good or Bad)
    **/
    quality: string;

    /**
     * Date when the value was read.
    **/
```

```
    timestamp: Date;

    /**
     * Describes the value type. E.g Int8, Int32, ComplexData.
    **/
    type: string;

    /**
     * Describes the object name For ComplexData types.
    **/
    name: string;

    /**
     * The model name if the data is modeled.
     **/
    model: string;

    /**
     * The name to the stage processing the event.
     **/
    stage: string;

    /**
     * Key:value object added to the event by previous stages in the pipeline.
    **/
    metadata: object;
}
```

System

The `System` variable contains read-only information about the hub. Its type is described below.

```
class System {
    /**
     * Read-only object containing all the System Variables defined for the hub instance.
     **/
    Variables: object;

    /**
     * Read-only object containing all the variable defined in the Environment (when enabled
     **/
    Environment: object;

    /**
     * Read-only internal values
     **/
    Internal : {
        DateTime: Date
    }
}
```

JavaScript | User Guide

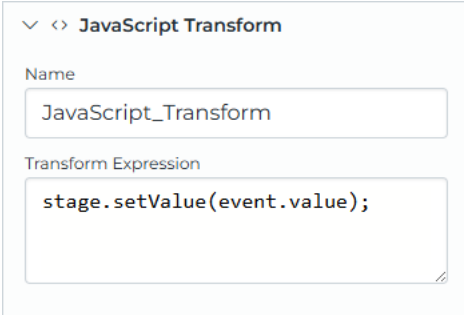
Source: https://guide.highbyte.com/configuration/pipeline/stages/transform/javascript/

JavaScript

Tags:

- Event.value
- Event.metadata

The JavaScript Transform Stage is used to configure advanced transformations using JavaScript. During execution the expression can be used modify the value, extract metadata from the value for use in other stages, and incorporate additional information into the event value.



Transform Expression

Required javascript expression. At a minimum the expression is expected to use `stage.setValue()` to specify the event value for future stages

Using Transform Expressions

The transform expression executes JavaScript to modify the event value and/or operate on metadata about the value that can be used in Output stages. Input references (using `{{}}`) in Pipeline expression fields is not currently supported. JavaScript in the transform expression has access to the following objects:

Event

Event

Stage

The `stage` variable has methods that can be used to control the state of the stage during transform evaluation.

```
class Stage {
    /**
     * The name of the stage
     **/
    name: string

    /**
     * The name of the pipeline
     **/
```

```
    pipelineName: string

    /**
     * The names of the stages the current stage outputs to
     **/
    outputs: string[]

    /**
     * Sets an event value. Expressions that use setValue multiple times will send multiple
     **/
    function setValue(value: any);

    /**
     * Sets the buffer key used for data buffering. This is applicable to buffer and filter
     **/
    function setBufferKey (key: string);

    /**
     * Sets the properties in a Write New stage. These are merged with properties in the sta
     **/
    function setWriteProperties (properties: object);

    /**
     * Sets a single property in a Write New stage. These are merged with properties in the
     **/
    function setWriteProperty (key: string, value: any);

    /**
     * Replaces the event metadata object sent to the next stage of the pipeline.
     **/
    function setAllMetadata(value: any);

    /**
     * Upsert metadata using the supplied key name. Setting a null/undefined value will clea
     **/
    function setMetadata(key: string, value: any);


    /**
     * Stops the pipeline from executing any further events down the current pipeline path
     **/
    function stop();
}
```

System

System

State

The `state` variable defines functions that can be used to access data that is persisted across pipeline executions and application restarts.

Below are some examples and the available methods. See Use Pipeline State for more examples.

```
    state.pipeline.set("myStateKey", 100);
    let myStateKeyValue = state.pipeline.get("myStateKey")  // value is 100
    let myUnknownKey = state.pipeline.get("unknown", 0)     // value is 0
```

```
class State {
    pipeline: {
        /**
         * Set a variable accessible from all stages. Setting a key value to null or undefin
         **/
        function set (key: string, value: any);

        /**
         * Get a variable accessible from all stages. The function returns undefined if the
         **/
        function get (key: string) : any | undefined;

        /**
         * Get a variable accessible from all stages or return the defaultValue when the var
         **/
        function get (key: string, defaultValue: any) : any;
    }
}
```

Example Transform Expressions

The following are examples of transform expressions.

Rename Properties in the Event

In this example assume the event payload contains OPC tags.

```
{
  "XD_R32_CONVERYOR8_INLET1_CV": 13.2,
  "XD_R32_CONVERYOR9_INLET1_CV": 12.9,
  "XD_R32_CONVERYOR10_INLET1_CV": 13.0
}
```

We want to rename these tags to add "PUMP" into the name between "CONVERYORX_INLET". The following transform performs this change.

```
let eventOut = {}

// Iterate each tag name
for (const tagName in event.value){
    // Split up to the 3rd "_"
    let tagNameParts = tagName.split("_");

    let newTagName = tagNameParts.slice(0,3).join("_") + "_PUMP_" + tagNameParts.slice(3,tag

    // Assign the new tagname the existing value
    eventOut[newTagName] = event.value[tagName]
}

// Set the value of the event
```

```
stage.setValue(eventOut);
```

The output is the following.

```
{
  "XD_R32_CONVERYOR8_PUMP_INLET1_CV": 13.2,
  "XD_R32_CONVERYOR9_PUMP_INLET1_CV": 12.9,
  "XD_R32_CONVERYOR10_PUMP_INLET1_CV": 13.0
}
```

Change Event based on State

In this example assume the event payload contains the state of the machine and we want to change the values based on the state. This could be an MQTT or SparkplugB payload.

```
{
    "line": "line1",
    "asset": "press1",
    "stateOn": true,
    "pressCount": 101,
    "lastCount": 98,
    "temp": 97.8
}
```

The following transform changes the values based on the machine on/off state.

```
let latestValue = event.value

if (!latestValue.stateOn){
    // Zero out the temp and signal no parts were made
    latestValue.temp = 0
    latestValue["newParts"] = 0
}else{
    // Calculate new parts
    latestValue["newParts"] = latestValue.pressCount - latestValue.lastCount;
}

// Set the value of the event
stage.setValue(latestValue);
```

Break Event into Multiple Events

Assume we have the following payload, and we want to break this up by assetID so that moving forward each event contains only events for an asset.

```
[
    {
    "assetId": 1,
    "value": 27
    },
    {
    "assetId": 2,
    "value": 55
```

```
    },
    {
    "assetId": 1,
    "value": 28
    }
]
```

The following transform will break up the array.

```
let dataOut = {}

// Iterate each asset
for (const assetValue of event.value){
    if (dataOut[assetValue.assetId] === undefined){
        // New assetId, create the array
        dataOut[assetValue.assetId] = []
    }

    // Add the value
    dataOut[assetValue.assetId].push(assetValue)
}

// Create an event for each unique assetId
for (const asset in dataOut){
    stage.setValue(dataOut[asset])
}
```

The first event has the following payload.

```
[
    {
    "assetId": 1,
    "value": 27
    },
    {
    "assetId": 1,
    "value": 28
    }
]
```

The second event has the following payload.

```
[
    {
    "assetId": 2,
    "value": 55
    }
]
```

Use Pipeline State

Assume we have the following payload, and we only want to run the pipeline when the `mode` value changes to `ERROR`

```
```

```
    {
    "assetId": "asset1",
    "value1": 27,
    "mode": "RUNNING" // possible values include "RUNNING", "ONBREAK", "ERROR"
    }
```

The following expression compares the event mode to the mode stored in the state and stops the pipeline unless the `mode` changes to `"ERROR"`

```
// Create a state key using asset id
let key = event.value.assetId + ".mode";
let newMode = event.value.mode;

// Load previous state using the current mode if mode has never been saved
let previousMode = state.pipeline.get(key, event.value.mode)

// Save new stage
state.pipeline.set(key, newMode);

// No change, stop
if (newMode === previousMode) {
    stage.stop();
    return;
}

// Not error, stop
if (newMode !== "ERROR") {
    stage.stop();
    return
}
```

Values stored in state using `state.pipeline.set` can be removed and edited from the State page.

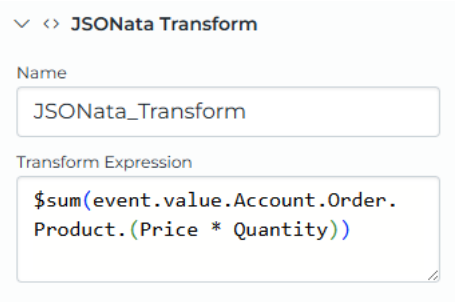
JSONata | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/stages/transform/jsonata/

JSONata

The JSONata Transform Stage is used to configure advanced transformations on incoming JSON using JSONata expressions. During execution, the expression can modify the event value, extract metadata for use in other stages, and incorporate additional information into the event data.

The full specification for JSONata can be found here.



Transform Expression

Required JSONata expression.

Using Transform Expressions

The transform expression executes JSONata to modify the event value and/or operate on metadata about the value that can be used in Output stages. Input references (using `{{}}`) in Pipeline expression fields is not currently supported. JSONata in the transform expression has access to the following objects:

Event

Event

System

System

Example Transform Expressions

The following are examples of transform expressions.

Aggregation Query Example

In this example, assume the event payload contains the following JSON.

```
{
  "Account": {
    "Account Name": "Firefly",
    "Order": [
      {
        "OrderID": "order103",
```

```
        "Product": [
          {
            "Product Name": "Bowler Hat",
            "Price": 34.45,
            "Quantity": 2
          },
          {
            "Product Name": "Trilby hat",
            "Price": 21.67,
            "Quantity": 1
          }
        ]
      },
      {
        "OrderID": "order104",
        "Product": [
          {
            "Product Name": "Bowler Hat",
            "Price": 34.45,
            "Quantity": 4
          },
          {
            "Product Name": "Cloak",
            "Price": 107.99,
            "Quantity": 1
          }
        ]
      }
    ]
  }
}
```

We want to calculate the total value of all products in all orders. The following transform performs this calculation.

```
$sum(event.value.Account.Order.Product.(Price * Quantity))
```

The output is the following.

```
{ "value": 336.36 }
```

Field-Mapping Example

In this example, assume the event payload contains the following JSON.

```
{
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Address": {
    "Street": "Hursley Park",
    "City": "Winchester",
    "Postcode": "SO21 2JN"
  },
  "Phone": [
```

```
    {
      "type": "home",
      "number": "0203 544 1234"
    },
    {
      "type": "office",
      "number": "01962 001234"
    },
    {
      "type": "mobile",
      "number": "077 7700 1234"
    }
  ],
  "Other": {
    "Over 18 ?": true,
    "Misc": null,
    "Alternative.Address": {
      "Street": "Brick Lane",
      "City": "London",
      "Postcode": "E1 6RF"
    }
  }
}
```

We want to perform some mapping to output the person's full name (First and Last Name) along with their mobile phone number. The following transform does this.

```
{
    "name": event.value.FirstName & " " & event.value.Surname,
    "mobile": event.value.Phone[type = "mobile"].number
}
```

The output is the following.

```
{
    "value": {
        "name": "Fred Smith",
        "mobile": "077 7700 1234"
    }
}
```

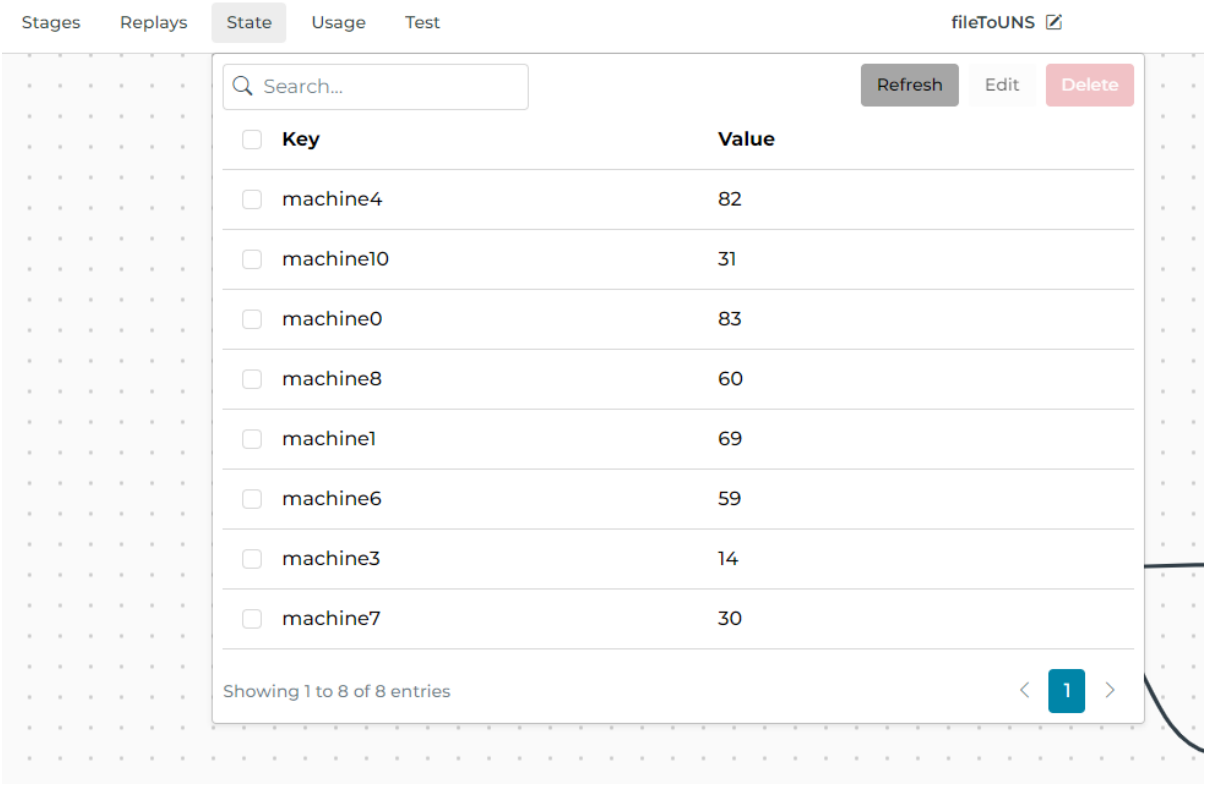
State | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/state/

State

Pipelines can contain state that is saved to disk between runs. Common examples of Pipeline state might be machine status, indexing of a data source (ex. SQL), and more. Pipeline state is accessible via the Transform Stage. See the State documentation for details on the syntax.

To view Pipeline state, select the State menu item in the Pipeline view. All keys and values for the current state are shown.



Remove State

To delete pipeline state select one/many rows from the list view, choose `Delete`, and confirm.

Edit State

To edit pipeline state select one/many rows from the list view, choose `Edit`, and confirm. The Edit modal is populated with the latest JSON value and can be modified by replacing the value with any valid JSON value.


Edit Metadata
Key
Value
machine4
82
machine10
31
Close
Submit

Triggers | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/triggers/

Triggers

Triggers push events into the pipeline and initiate a data flows. For example, a poll trigger may fire every minute and initiate the Pipeline to then read an Instance. Pipelines can have one or more triggers. Each trigger can be independently enabled or disabled for troubleshooting pipelines.

Common Configuration

Name

The name of the trigger. This name is used in replay and tracking, and must be unique in the pipeline.

Enabled

When disabled the trigger does nothing, and produces no events. When enabled the trigger may produce events based on its configuration.

Trigger Events

Trigger events have a value and metadata. The value and metadata is different based on the trigger type.

See the following links for more information on the available trigger types.

- API
- Callable
- Cron
- Event
- Flow
- Polled

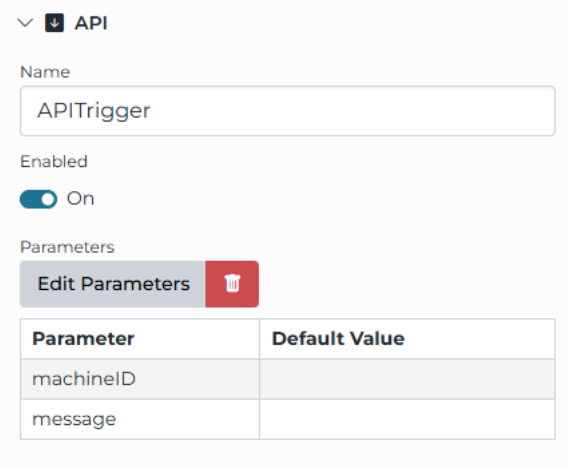
API | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/triggers/api/

API

API Triggers expose a pipeline to the REST Data Server. For examples on how this works, see the REST Data Server page. API Triggers support parameters, allowing you to scope what can be sent to the pipeline.

Configuration



Enabled

The API Trigger must be enabled for it to be visible to the REST Data Server.

Parameters

Parameters allow users to define the expected payload structure that a pipeline can accept over the API. Each parameter includes a name, type, and optional default value. When parameters are defined, incoming requests must include a JSON payload that matches this structure. Parameters without default values must be explicitly provided in the payload. If the payload doesn't match the defined parameters, the trigger fails the request.

If no parameters are defined, the pipeline will accept all incoming payloads. Requests that are of JSON content types will be treated as JSON values, while non-JSON content types will be treated as string values.

Custom Pipeline Behavior

Overriding Default HTTP Status Codes

By default, requests handled by the REST Data Server return predefined status codes with the generated HTTP responses. To override these, users can use pipeline metadata to specify a custom status code. To leverage status code overrides, the pipeline must produce a valid response using the **Return Stage** without completing in an error state.

Set the `HTTPResponse.statusCode` key to your desired status code. For example, to return a 202 Accepted response:

```
stage.setMetadata(
    "HTTPResponse",
    {
```

```
        "statusCode": 202
    }
);
```

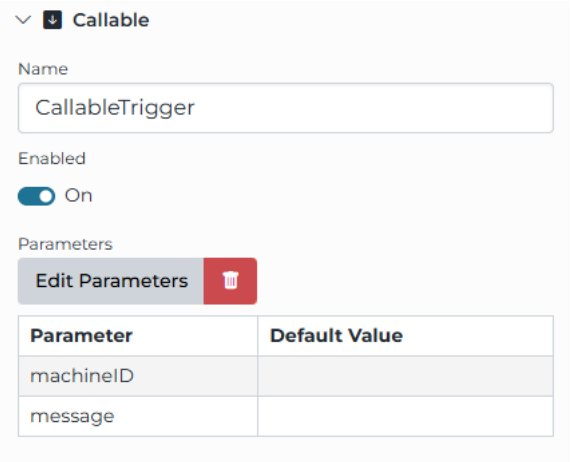
Callable | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/triggers/callable/

Callable

Callable Triggers allow a pipeline to be called internally, either through a Namespace or via a Sub-pipeline stage.

Configuration



Enabled

The Callable Trigger must be enabled for it to be called.

Parameters

Parameters allow users to define the expected payload structure that a pipeline can accept. See the API Trigger parameters section for more details.

Cron | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/triggers/cron/

Cron

Cron Triggers fire at a specific time based on a Cron string. For example, the Trigger can be configured to fire every Monday at 8am. When enabled, the trigger uses the system clock and Cron string to determine when to fire. The trigger fires empty events, and includes the time of the event in the metadata.

Configuration

Schedule

Cron schedules use system clock time and are subject to time zone changes. Unix-style cron syntax is supported. The general form of the cron string is `minute hour day month day-of-week`. Below are some examples.

- `0 0 * * *`
  - at midnight every day
- `* * * * *`
  - triggers every minute.

Timezone

Configures the schedule to use UTC or local time.

Logging

Choices include **None** and **Schedule Events**. When set to **Schedule Events** the trigger will log an information message each time a new trigger time is calculated.

Event Values

The event value and metadata are both empty. To access when the event occurred in a pipeline use the `event.timestamp` property.

Event | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/triggers/event/

Event

Event Triggers subscribe to a single event source. When the source produces an event (ex. MQTT topic) the event enters the Pipeline. If the source is invalid, or does not generate events, the trigger goes into an error state.

Configuration

Reference

A reference to the source of the events. For example, this could be an MQTT input or an OPC UA tag.

Event Values

Value


```
{
  "attr1": "value of the event",
  "attr2": 2
}
```


Metadata


```
{
  "source": "Connection.mqtt.out",
  "timestamp": "2024-06-24T13:54:07.606Z"
}
```

Flow | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/triggers/flow/

Flow

Flow triggers can operate in event or polled mode, and read one or more sources, perform trigger logic, and trigger events into the pipeline.

Flow triggers are used to migrate Flows from version 3.X and older to version 4.0 and function the same as in version 3.X. The new Polled and Event, combined with Read stages may be better suited for new projects.

Configuration

References

A list of sources to read data from when the flow executes. Each source generates a single event into the pipeline. Drag & drop sources from the reference panel. All reads for each source are issued at the same time, and the flow waits for all reads to complete before outputting events.

Sources can be templated. For example, if a templated Instance expands into 10 Instances, referencing the templated Instance is the same as manually creating and adding all 10 Instances.

Sources can also take parameters (e.g., `{{Connection.SQL.machines({"id":"3"})}}`). See the Input and Instance Parameters subsection under Templating for more details on templating and parameterizing sources.

Execution Type

Specifies the type of Trigger. Flows can trigger in Polled or Event mode.

Type	Description
Polled	Operates on an interval. On each interval the flow evaluates the trigger mode and expression, and when true the flow reads the source references.
Event	Event based flows execute when a new event occurs and guarantees each event is processed. For example, an event based flow can be triggered every time an MQTT message arrives. If 100 messages arrive in a second, each message is queued and processed in order.

Interval

When the Execution Type is Polled, this specifies how often the flow is processed. This setting it not shown in Event mode.

Mode

Specifies when the flow triggers.

Mode	Description
Always	For Polled mode, the flow executes on every interval. For Event mode, the flow triggers for every event received.
On change	On the first Poll or Event, the expression is evaluated, the result is stored, and the flow executes. On successive intervals, the expression is evaluated and compared with the previously stored result. If a change is detected, the flow executes, otherwise no action is taken.

Mode	Description
On true	The flow executes every time the expression transitions from false to true (rising edge).
While true	The flow executes as long as the expression remains true.

Expression

In Polled mode this field represents the JavaScript logic performed to detect On change, On true, etc. The field can reference one or more sources.

In Event mode, this field represents the event source(s) to trigger off of when the mode is set to Always. In On change and other modes, this field represents the trigger sources and logic.

Trigger Delay

When the flow is in Event mode or is in Polled mode with a trigger Expression, this is the amount of time the flow waits from the Event Source / Expression evaluation to reading source data. This can be useful in cases where there is a delay between getting an event and when the source should be read. An example of this would be a machine where an OPC UA machine status tag changes, but requires a delay to ensure the remaining OPC UA tags are updated.

Publish Mode

Controls what is published per source.

Publish Mode	Description
All	The event contains all source data values.
Only changes	The event contains only source data values that have changed from the previous interval. For modeled instances, all attribute values are written.
Only changes, compressed	The event contains only source values that have changes from the previous interval, and only the changed values. For modeled instances, only the attribute values that changed are written.

Only Changes, compressed Example

Assume the source below is published on the first flow execution.

```
{
  "attribute1": 1,
  "attribute2": 2,
  "attribute3": 3
}
```

On the second flow execution only the value for `attribute2` changes to 20. The other attributes stay the same. The following is published.

```
{
  "attribute2": 20
}
```

Flow Templating

Flows can be templated like Inputs and Instances. Templating a flow accelerates the virtual creation of many flows that have the same basic configuration. Template settings can be used in sources, targets, and trigger expressions.

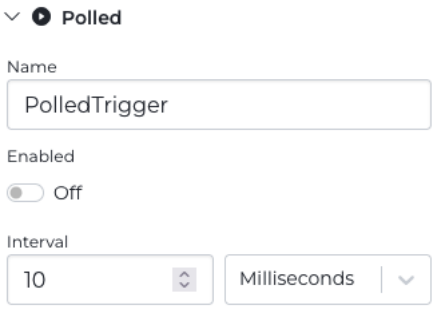
See Templating for more details.

Polled | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/triggers/polled/

Polled

Polled Triggers fire on a timer. For example, the Trigger can fire every minute. When enabled, the trigger fires its first event when the Pipeline starts, and then based on the Interval setting. The Polled trigger fires empty events, and includes the time of the event in the metadata.

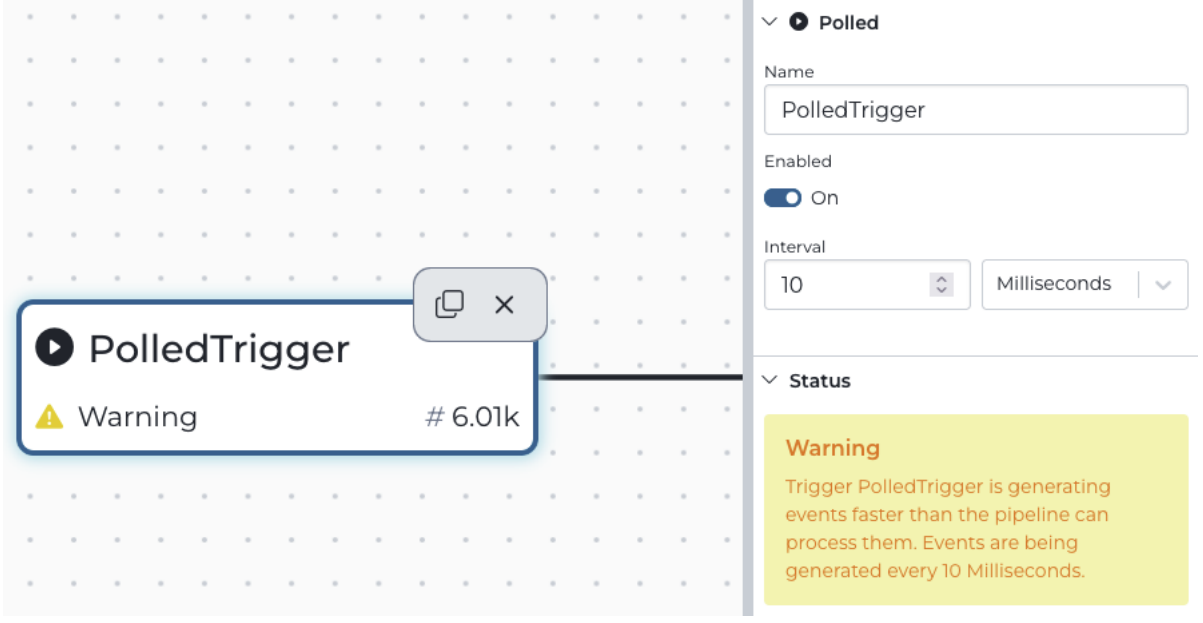


Configuration

Interval

The amount of time elapsed between firing events. The trigger fires immediately when the pipeline starts and every interval thereafter.

If the amount of time it takes to process the pipeline event is greater than this interval then the next event will be delayed. For example, if a Polled Trigger fires every 1 second but it takes 5 seconds to process the generated event, then the trigger will only fire events every 5 seconds. In the event the Polled Trigger cannot fire events as fast as the specified interval, a warning will be displayed on the Polled Trigger indicating that it is generating events faster than the pipeline can process them.



Event Values

The event value and metadata are both empty. To access when the event occurred in a pipeline use the `event.timestamp` property.

Troubleshoot | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/troubleshoot/

Troubleshoot

Pipelines support pipeline and stage level statistics, as well as debug mode and event replay to help troubleshoot the logic and performance of the pipeline.

- Debug
- Error Handling
- Replay
- Statistics

Debug | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/troubleshoot/debug/

Debug

I/O stages may produce side-effects while using debug mode. Click here for more information

Debugging

Debugging is a powerful tool that provides a safe environment to inspect, update, and test a pipeline.

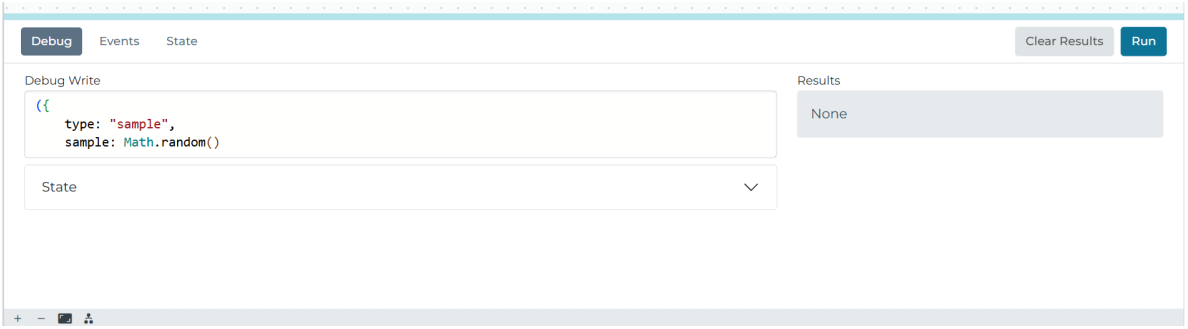
How to Debug a Pipeline

First, click on the bug icon in the top left corner of the screen to show the debug tools.



The debug tools consist of three sections, and two buttons (Clear Results, and Run)

Section	Purpose
Debug	Used to specify the information that will be used to execute the debug run
Events	A side-by-side diff of the event
State	A side-by-side diff of the state



Performing a debug run

To perform a debug run:

1. Start in the Debug section.
2. Enter a value in to the “Debug Write” field.
    - This value is used as if it was an `event.value` emitted by a trigger (Or by performing a Test Write).
3. Click the “Run” button
4. If an error occurred that caused the debug run to fail it will appear in the “Results” section of the Debug panel
    - This only refers to errors such as a misconfigured “Debug Write” value for example that causes the whole debug run to fail.
    - This does not apply to stage errors which can be viewed by selecting the stage.

At this point you will notice that all of the status/statistics in the graph are replaced with the status/statistics of the debug run.

To view the real status/statistics again and start over, click the “Clear Results” button

Debugging Changes

The pipeline does not have to be saved to perform a debug run.

When the "Run" button is clicked the debug run will be executed using the current unsaved state of the pipeline that you're working on.

This means that you can add stages, move stages around, and change settings on stages; and all of those changes will be included in the debug run.

Viewing Events

The Events panel shows a side-by-side diff comparing all of the incoming events (left), to outgoing events (right).

Some stages may produce more than one event (the breakup stage for example) in that case the paginator above the incoming and outgoing events can be used to click through the events that the stage processed.

Viewing State

The state of a pipeline is stored as an array of key/value objects.

If the pipeline is supplied some state, then that state will appear in this section.

Debugging a Pipeline that uses State

Expanding the "State" section in the Debug section provides a handful of actions and two fields that can be used to configure state when debugging a pipeline.

Filling in these fields manually is not recommended. See (Get Production) below

- The "State" field is an array of key/value objects that represent the State of this pipeline
- The "Dependency State" field is an object, where each property is the name of a subpipeline and it's array of key/value State objects

Get Production - This button will automatically populate both fields with the current State of this pipeline (and all of it's Subpipelines)

Get Debug Result - After performing a debug run the state can be cycled by pressing the "Get Debug Result" button which will replace the state in both fields with the results from the previous debug run.

Clear - Use the clear button to reset both fields

More Information

Is debug mode safe?

Debugging a Pipeline simulates a pipeline execution in a safe environment that will not impact the saved pipeline in any way.

- The State, Status, and Statistics of the pipeline will not be impacted
- Any Subpipelines will also be executed as a debug run, meaning all of the "safe environment" information here also applies to Subpipelines
- Inputs, instances, and namespaces are read as Test Reads

Limitations - Debug runs can cause some side effects

- The Write and Write New stages will still write data
- This is also true for Write and Write New stages nested in a Subpipeline

What does the Debug Write field support?

Supported values

- Expressions
- Primitive values: String, number, boolean

- Objects - *must be wrapped in parenthesis*
- Arrays

Unsupported values

- JavaScript
- null
- undefined

Note: Leaving Debug Write empty will default to an empty object `{}`

Unsupported stages

The following stages are currently unsupported in debug mode and will simply pass the event unchanged to the next stage in a debug run.

- Size Buffer
- Time Buffer
- On Change

Error Handling | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/troubleshoot/error_handling/

Error Handling

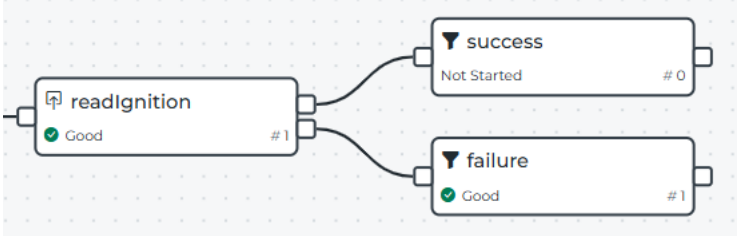
Pipeline Errors

Errors can occur either in the overall pipeline or in pipeline stages. Example errors could include exceeding the maximum inbound event queue size or individual stage errors like bad reads/writes or incorrect stage logic.

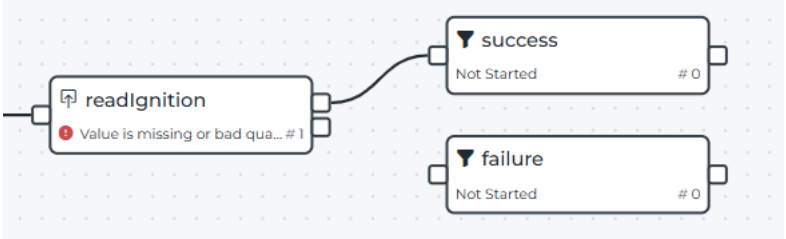
The pipeline is in error if any of the stages have experienced an error. Errors are also sticky, meaning that if an error only occurs on the 10th event processed by the pipeline, but the 11th event succeeds, the pipeline is still in an error state until you resave the pipeline or click the stage that had the error and clear it. This way intermittent errors are not lost. To clear an individual error, click the stage that's in error and click "Dismiss", shown below.

Stage Errors

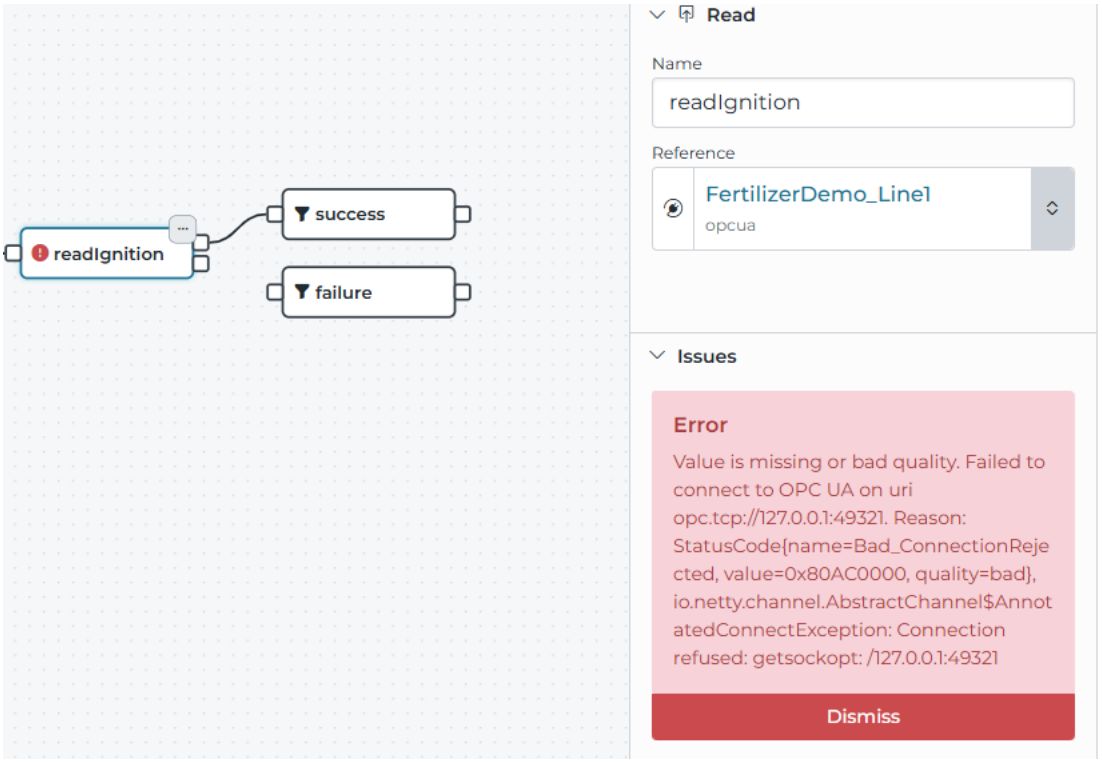
Some stages, like read, write, etc have success failure paths as part of the stage configuration. This allows you to handle errors inside the Pipeline without putting the pipeline in error. For example, here is what happens when the failure path is connected to the read stage and the read fails. You can see the event takes the failure path give the execution count.



If the failure path is not connected, the stage and pipeline go into error.



To clear a stage error, click the stage and select "Dismiss" underneath the error message.



Catching Pipeline Errors

There are cases when you want to standardize the way pipeline errors are caught to send them to an internal ticket management system for example.

To do this you can set the **Error Handler** setting in the pipeline to External (defaults to None) and select a pipeline to send errors to. In this mode the pipeline continues to behave and reflect errors like before, but error events are sent to another pipeline.

The event sent to the external pipeline for errors looks as follows

```
{
   value: any, // Event value
   metadata: { // Metadata
      pipelineMetadata: { // Metadata related to the incoming event value

      },
      pipelineError: { // Metadata regarding stage failure
         name: string, // Name of pipeline that errored out
         stage: string, // Name of stage that errored out
         type: string, // Type of stage that errored out
         timestamp: string, // Time the error occurred
         error: string // Error message
      }
   }
}
```

Replay | User Guide

> Source: https://guide.highbyte.com/configuration/pipeline/troubleshoot/replay/

Replay

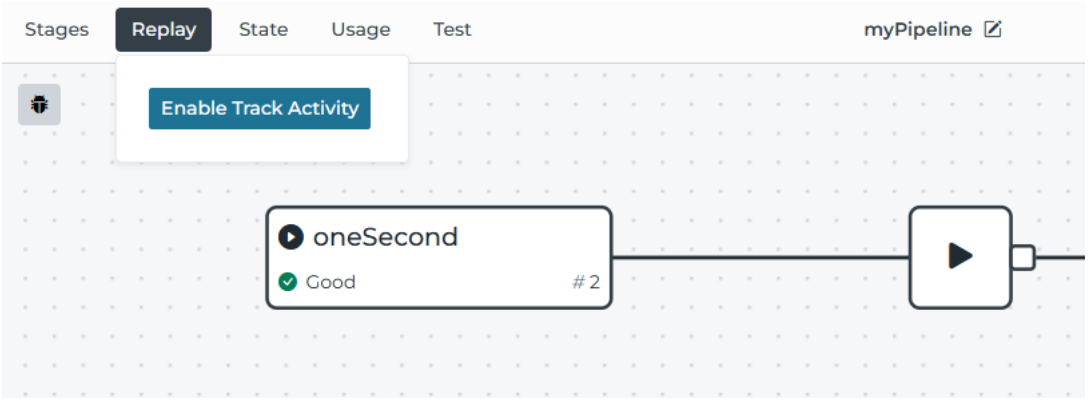
Leaving replay on for a pipeline in production can degrade pipeline performance as each stage requires disk access to get/save tracked events. We recommend only using this feature for debugging.

Replay Mode

Replay Mode provides detailed information about previous pipeline executions. From the Replay List, recent pipeline executions can be replayed, and individual events can be inspected in detail.

Recording Replays

To start recording Replays, click `Enable Track Activity` under Replay in the Pipeline menu.



When `Track Activity` is enabled, the last 100 pipeline executions will be stored on disk in a database (`intelligencehub-pipeline-activity.db`) located in the appData directory. After the 100 replay limit is reached, the oldest replay will be deleted. The message "This replay has expired" will be shown when attempting to access a replay that has been deleted.

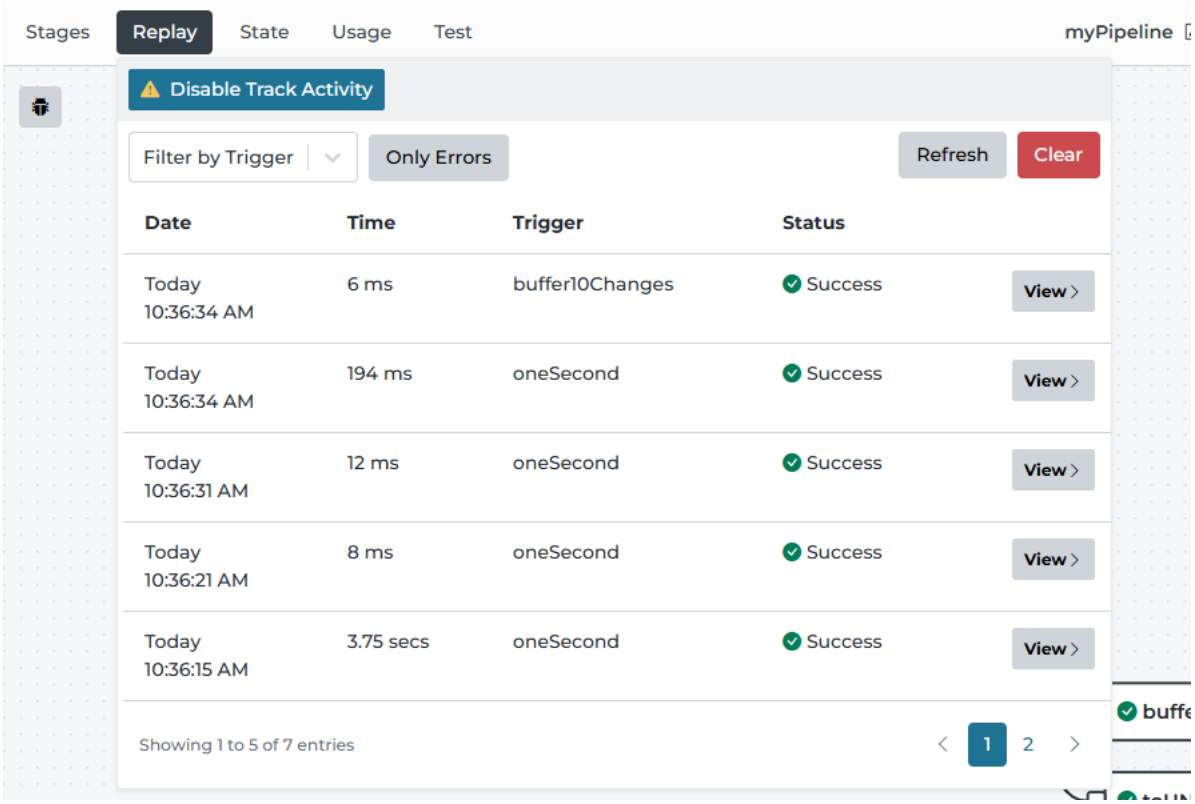
The replay list can be paused by disabling "Track Activity". This will not delete any saved replays, but will pause the recording of new replays. Note the timestamp shown for a replay is the time when the pipeline finished processing the event.

Entering Replay Mode

To enter replay mode, click the Replay tool to open the Replay List, then select a replay from the list. Replays can be triggered by:

- Triggers
- Test Writes
- Buffer Stages
    - *(e.g.) When a Time Buffer expires it will re-trigger the pipeline from that stage.*

The name of the trigger that the event originated from is shown in the Trigger column.



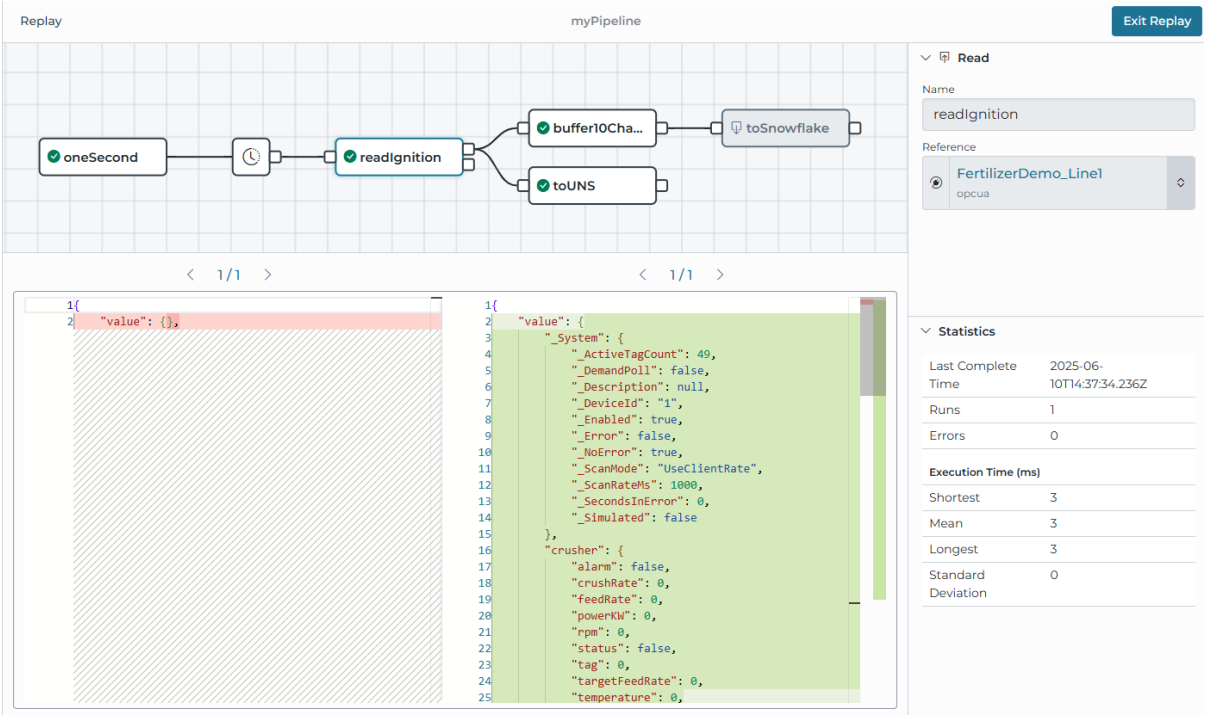
When a replay is selected, the view will change to “Replay Mode”.

Using Replay Mode

When in Replay Mode the pipeline graph cannot be edited, and will reflect the exact state of the pipeline when the replay was recorded.

Stages that ran successfully during the run are shown with a green status, errors are red, and stages that are grayed did not receive any events.

Selecting a stage in replay mode provides more detail, including the event data going into and out of the stage, as well as the stage configuration, status, and statistics.



Exiting Replay Mode

Replay mode can be exited by using the “Exit Replay” button in the top center of the graph, or by clicking the Replay Tool and selecting “Exit Replay” on the details card for the selected transaction.

Deleting Replays

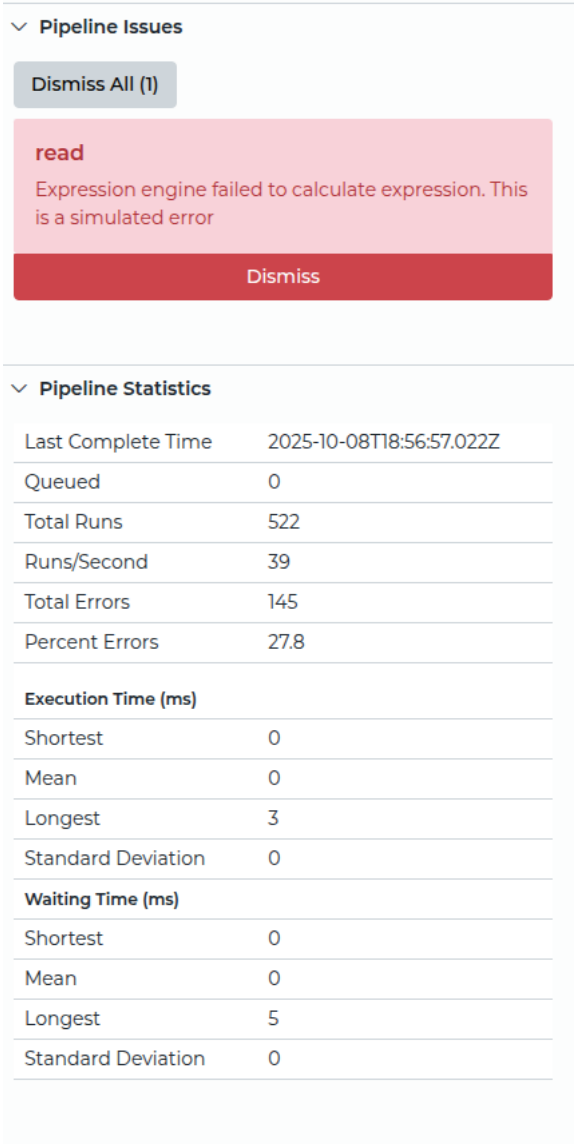
Using the “Clear” button in the Replay List will permanently delete all recorded replays for that pipeline.

Statistics | User Guide

Source: https://guide.highbyte.com/configuration/pipeline/troubleshoot/statistics/

Statistics

To see Pipeline Statistics, click on the Pipeline start stage. For information on the available metrics see the Pipeline/Stage Statistics tables on the Status and Metrics page.



Servers | User Guide

> Source: https://guide.highbyte.com/configuration/servers/

Servers

The Intelligence Hub supports a REST Data Server, MCP Server, and MQTT Broker. These must be explicitly enabled in the application settings.

This section covers the specific setup, features, and use cases supported by these servers.

- REST Data Server
- MCP Services

MCP Services | User Guide

Source: https://guide.highbyte.com/configuration/servers/mcp/

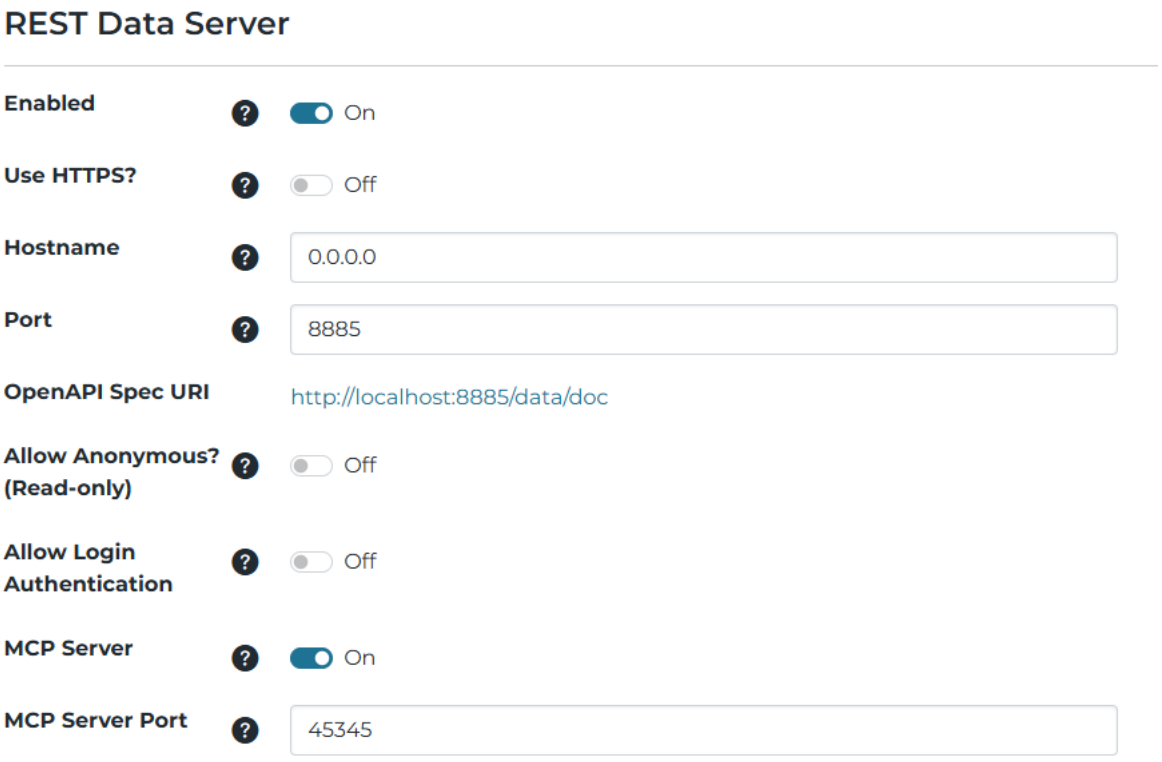
MCP Services

The Intelligence Hub acts as an MCP Server, exposing MCP Services that AI Agents can use to access industrial data, execute operations and perform configuration and troubleshooting. This section covers how to enable MCP, and the different MCP Tools the hub exposes.

Enable the MCP Server

The MCP server is enabled under the REST Data Server options. To enable MCP do the following:

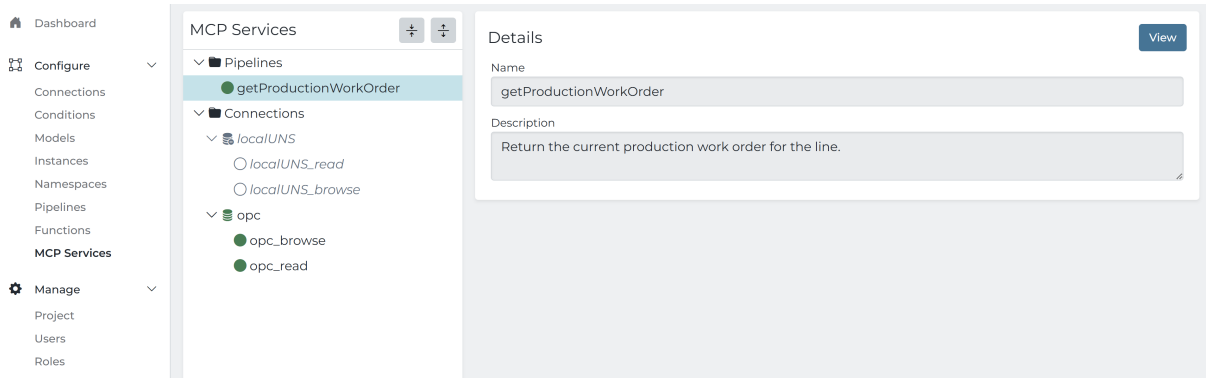
1. Enable the REST Data Server
2. Enable the MCP Server



The MCP server supports the Streamable HTTP protocol, and adheres to the 2025-03-26 revision of the MCP specification.

MCP Services

The MCP Services tab provides a view of all the MCP Tools configured in the project. Tools that are active and enabled are shown in green. Tools that are disabled are shown as grayed out. Clicking a tool shows the tool description. Clicking the View button on a tool takes you to the project configuration for the tool.



Pipelines

Pipelines allow you to build custom MCP tools that can access industrial data or perform operations. Each Pipeline with an API Trigger is shown as a tool under Pipelines in the MCP Services view. The pipeline name, description, and parameters are included as context to the AI Agent.

In this way, think of the pipeline description field as part of prompt engineering, where you're instructing the LLM when and how to best use a tool. As an example, a tool description like "call this to create a work order" is very little context. But something more verbose like "call this to create a maintenance work order when you think work needs to be performed on a machine. Make sure you always include the machineID of the machine that needs the work and a verbose message on what work needs to be done and why you think it needs to be done" will likely yield better results. Prompt engineering is not a science, but generally the more context the better.

In addition to the API Trigger, Pipelines must also include a return stage to return a response to the AI Agent during a tool call.

Connections

Currently OPC UA, MQTT support out-of-the-box tools for basic data discovery and read-only operations. These tools are generally not meant to be used in production, as they provide broad and read-only access to the data sources. However they are useful when using a chat bot or AI Agent to ask questions of the data. To enable these tools, navigate to the Connection Settings for the connection and enable the `Enable MCP Tools` option.

The MCP Client Connection also supports the `Enable MCP Tools` option. When enable this connection includes all the tools provided by the underlying MCP Server it's connected to.

Configuration

Configuration tools are currently not shown in the MCP Services tree. These tools can be enabled by going to the REST Data Server and enabling the `MCP Configuration Tool` option.

The current release supports tools for creating and editing Pipelines, including tools to get the Pipeline JSON Schema, get example Pipelines, and Create, Read, and Edit Pipelines. These tools allow AI Agents to build Pipelines from human language. A few things to note when using these tools:

1. Currently the tools don't know about Connections, Inputs, and Outputs. You must provide the Connection and Input name for example when asking to read a data source
2. The tools don't know about output settings. For example, when asking for the Pipeline to write to S3 with a given bucket name, you need to provide an example of the output settings/JSON for the tool to create a correct Write New stage

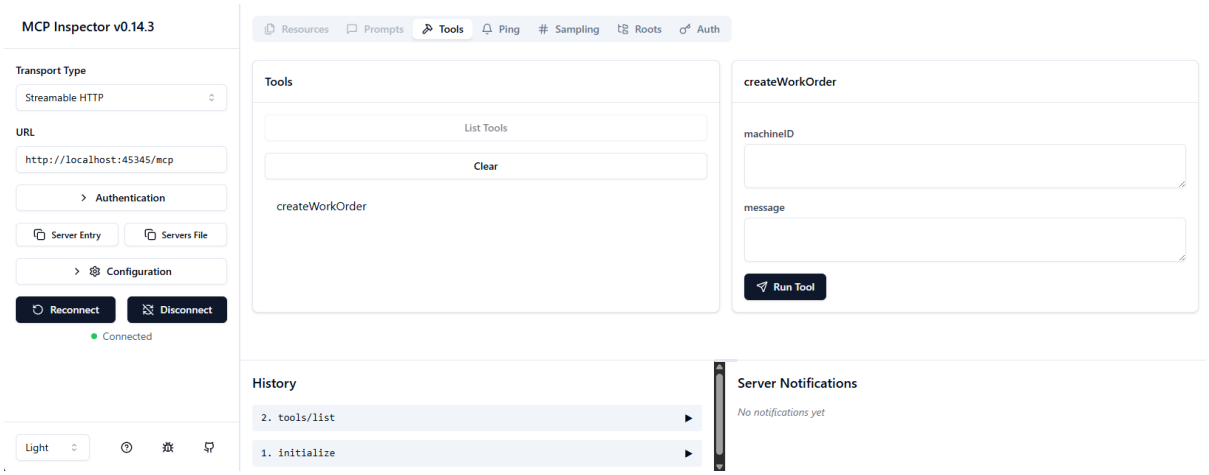
In general these tools get you 60-80% of the way to building the Pipeline, and are good for making edits to existing Pipelines. We'll continue to enhance these tools and provide additional context in future releases.

Connecting MCP Inspector

MCP Inspector is an open source tool used to connect to an MCP Server and view tools. To test using MCP Inspector do the following:

1. Follow the instructions here to launch MCP Inspector https://modelcontextprotocol.io/docs/tools/inspector
2. Change the Transport Type to Streamable HTTP

3. Change the URL port to 45345 (default port for our MCP server)
4. Append `/mcp` to the URL, after the port
5. Expand Authentication and copy in the bearer token created (see REST Data Server) for details on creating the token
6. Click Connect
7. Click List Tools
8. You should see all the pipelines with API Triggers enabled and the parameters for each API Trigger



Tool Refresh

For now, the MCP server refreshes it's tools list any time a client requests new tools. You can generally force this by one of the following:

1. Restart the MCP server (toggle off/on). This will force clients to reconnect and re-request tools
2. Restart the MCP Client (or agent)
3. Prompt the agent/LLM to refresh

MCP does support a way to notify clients of tool changes, but this currently isn't supported given that most clients don't support it either. In general, tool changes only occur during development, and should be rare in production workflows.

Access Control

The MCP Server uses the existing **roles and claims** system to manage agent access.

The following table lists the MCP claim components required for each type of tool category. Use these when assigning roles/claims to users or API keys.

Tool Category	Resource	Actions
Pipeline	pipeline	read, execute write
Connection	connection	read, execute read, execute write
Config	pipeline	read, create, update

User and API Key Setup

A new user can be created and assigned the appropriate **roles** and **claims** for MCP access. An API Key can be generated for this user, which can be used as a Bearer token to authenticate requests to the MCP Server. See REST Data Server for details on creating the API Key.

Additional information on managing users, roles, and claims is available in the Users and Roles documentation.

Example

Suppose you want an agent to only have access to a specific MQTT Connection you configured. Tag-based access control enables this restriction:

1. Create a tag (see Tagging for details) and assign it to the specific MQTT connection
2. Define a role with a claim that specifies the required resources, actions, and tags
3. Assign the role to a user
4. Generate an API key for the user and use it as a Bearer Token for MCP Client authentication against the HighByte MCP Server



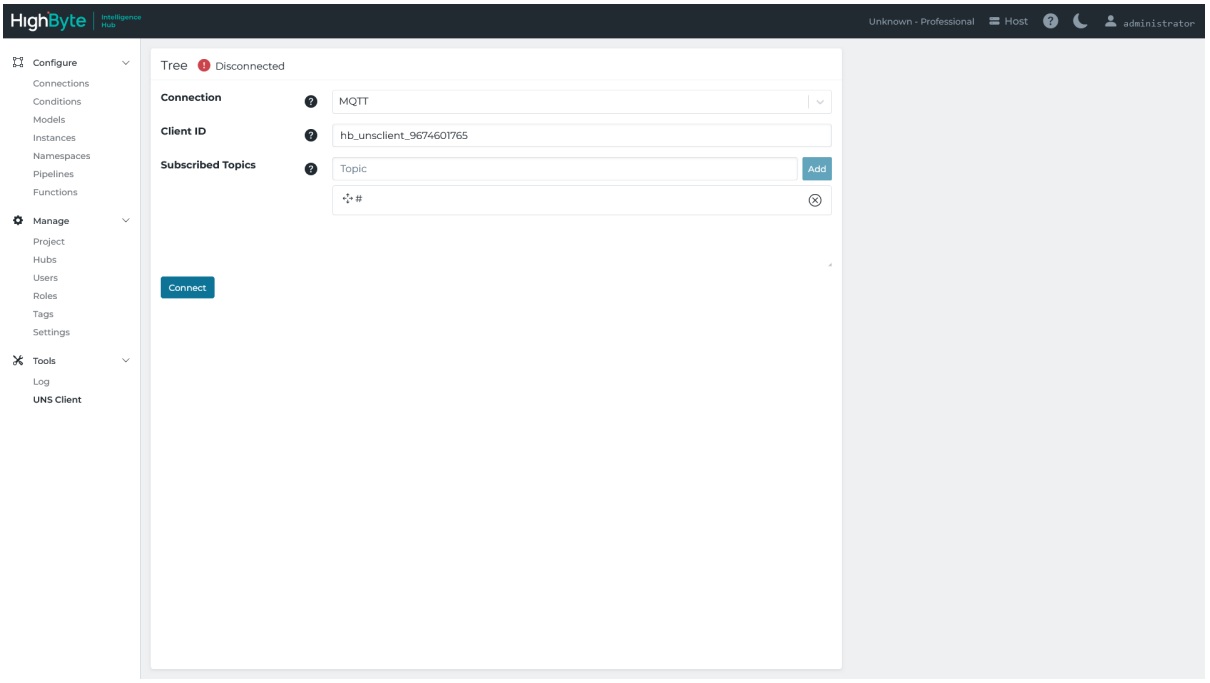
UNS Client | User Guide

> Source: https://guide.highbyte.com/configuration/uns_client/

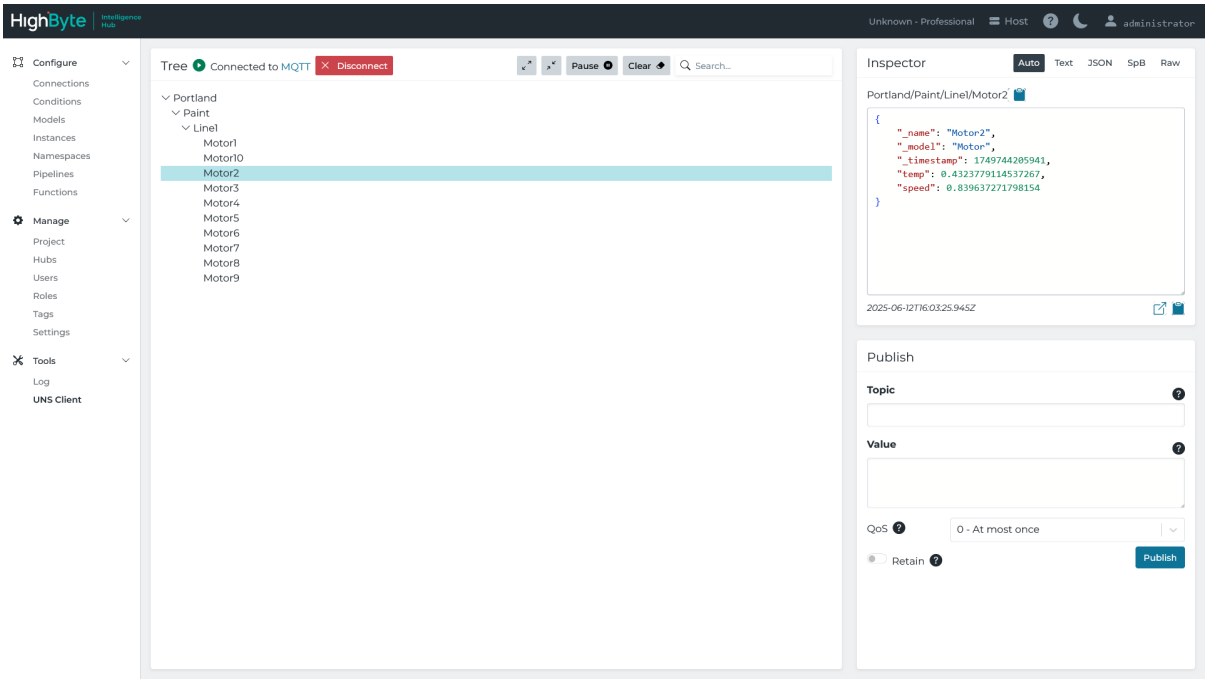
UNS Client

The UNS Client allows you to connect to any MQTT broker and view the data either in JSON or Sparkplug B format. It's a quick and easy way to visualize your UNS while you're building it.

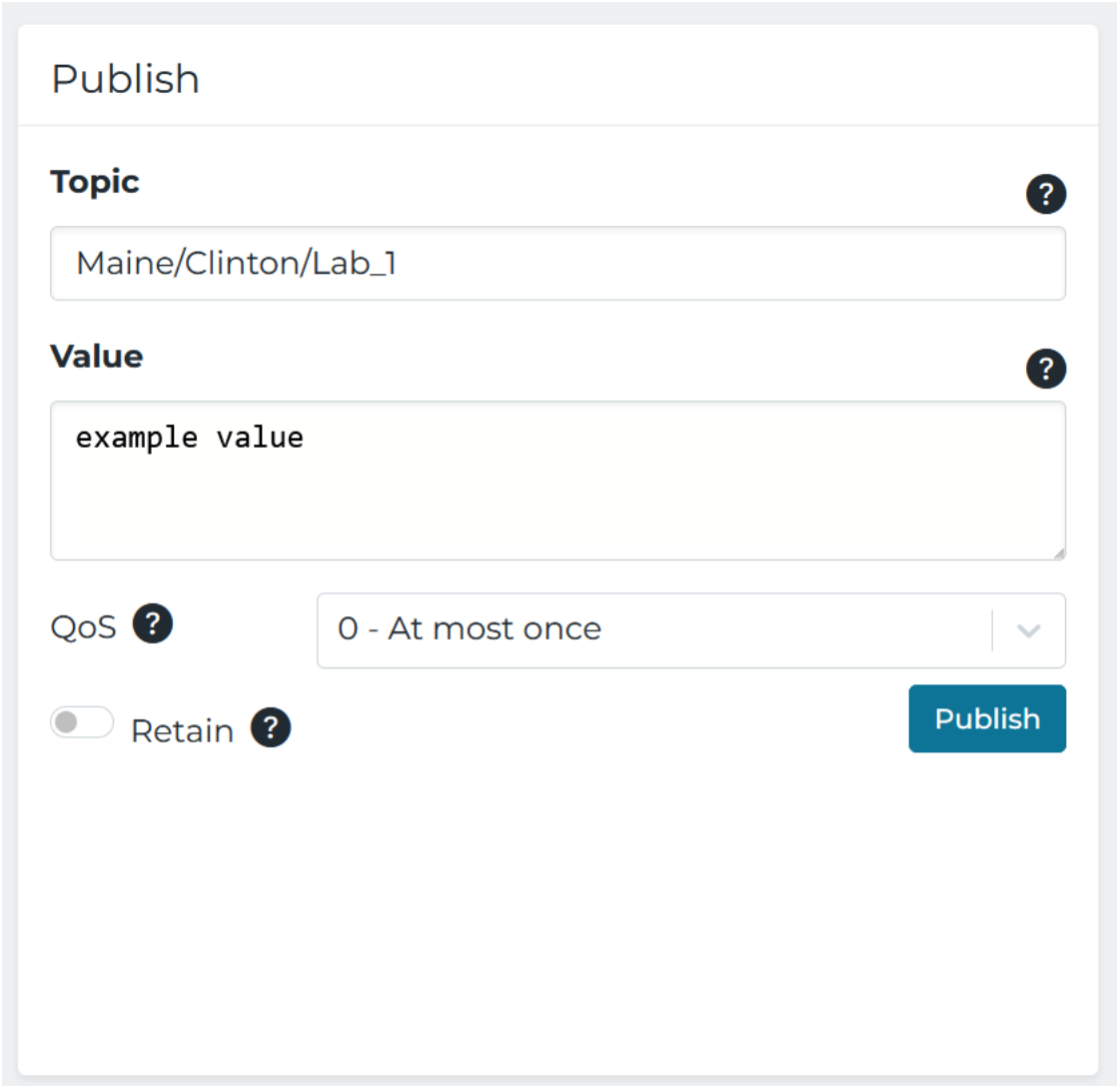
To use the UNS Client, choose an MQTT or Sparkplug B connection from the Connection dropdown. Optionally change the list of topics to subscribe to. By default the client subscribes to `#`.



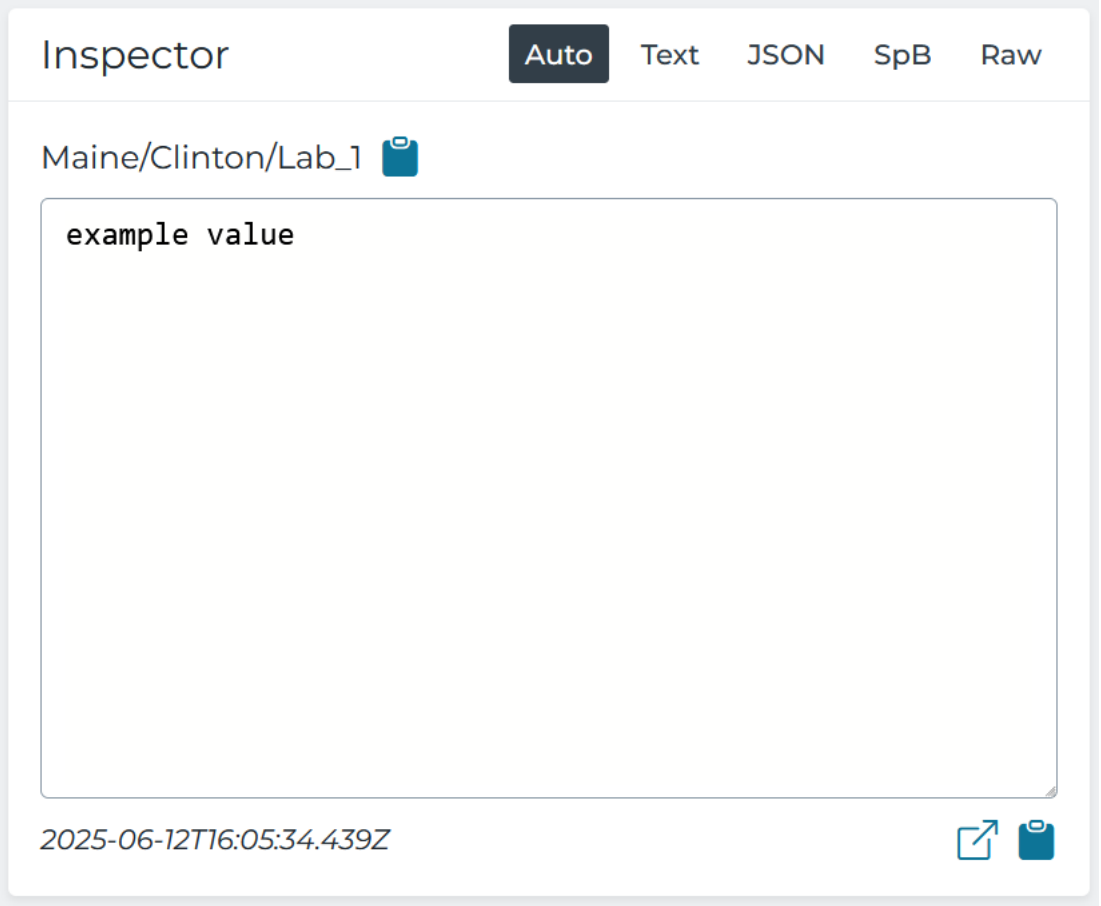
Once connected the UNS Client renders all MQTT message topics as a tree in the inspector pane. Topics can be expanded. When a topic with data is selected, the data is shown in the inspector pane. The UNS Client attempts to detect the message encoding and decode the message accordingly.



The client also supports publishing MQTT messages using the publish pane. Optionally set the Quality of Service (QoS) level and Retain flag.

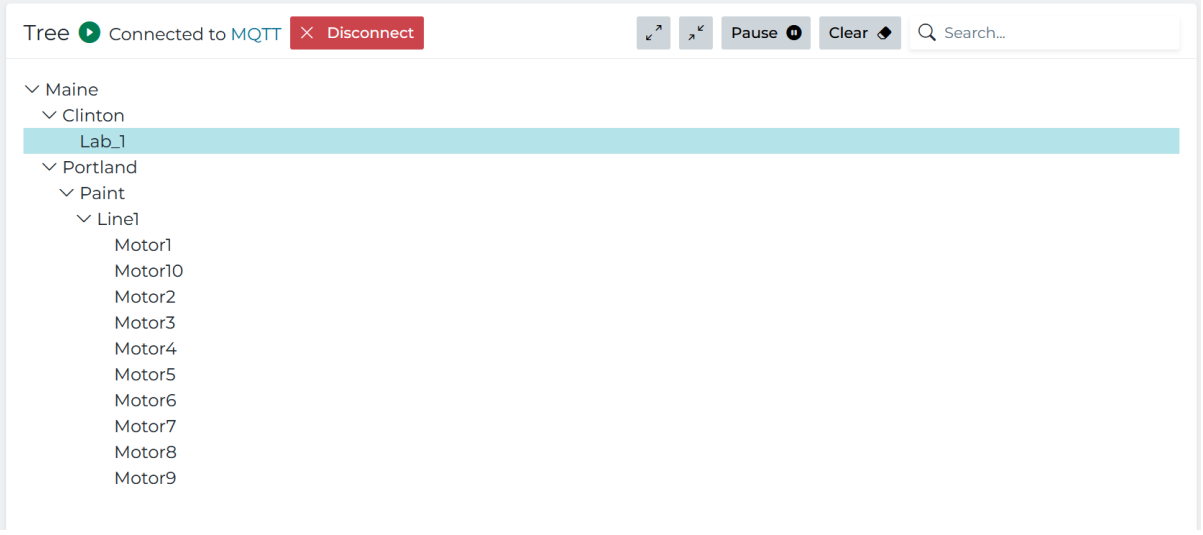


The Inspector pane defaults to automatic decoding detection, but allows users to override the decoding with the options for Text, JSON, Sparkplug B and Raw.



The Tree toolbar has a number of options; from left to right: Disconnect, Pause, Expand All, Collapse All, Clear and Filter

- Disconnect - disconnects the UNS Client from the broker and allows for changing which connection and what topics are used.
- Pause - Pauses updates to allow for inspecting fast changing messages.
- Expand All - to show all nested topics in the tree.
- Collapse All - to hide all nested topics in the tree.
- Clear - To remove all topics from the tree and only populate when new data is received.
- Filter - Filter all the topics based on a case sensitive exact match search. (eg. the topic `Maine/Portland/PaperMachine_1/Motor_1` would match `Motor_1` but would not match `paper`)



Setup | User Guide

> Source: https://guide.highbyte.com/setup/

Setup

HighByte Intelligence Hub has minimal System Requirements and can be installed on Windows, Mac and Linux. We also offer a docker container for rapid testing or production deployments.

- Application Settings
- Command Line Options
- Docker
- Git Integration
  - Git Example
- Installation
  - Installing Java
  - Licensing
  - Application State
- High Availability
- OpenTelemetry
- Redundancy
- System Requirements
- Upgrade
  - Upgrading from 3.4 to 4.0

Application Settings | User Guide

> Source: https://guide.highbyte.com/setup/application_settings/

Application Settings

Application settings are configured through the UI in the `Admin | Settings` section, or may be manually edited using the `runtime\intelligencehub-settings.json` file. By default, the installation includes an example file with a `.template` extension. This file contains default settings, but will not be used by the runtime until it is renamed.

Application Settings are loaded on startup. Changes made through the UI page take effect immediately. Any manual edits made directly to the file will not take effect until the runtime is restarted.

By default the runtime looks for the settings file in the appData directory. The location of the settings file can be configured when launching the runtime using a java property or environment variable. Below are examples of how to change the location of the file.

```
# As a java property
java -cp intelligencehub-X.X.jar:/lib* -Dhighbyte.settingsFile=/path/to/settings.json start

# As an environment variable
HIGHBYTE_SETTINGS_FILE=/path/to/settings.json java -cp intelligencehub-3.0.jar:/lib* start
```

Below is a list of application settings and associated descriptions that are available in the file.

Directories

Setting	Description
directories.appData	Specifies the location for loading and saving configuration, event log, user management, identity provider settings, and other application data files. If this setting is not specified or is invalid, application data files will be stored in the appData directory. Note: Specifying a Windows path requires any path delimiters (`\`) to be properly escaped. For example, a path to `C:\Program Data\HighByte\Intelligence Hub` should be set as `C:\\ProgramData\\HighByte\\Intelligence Hub`.
directories.storeForwardData	Specifies the location for loading and saving store and forward data files. If this setting is not specified or is invalid, store and forward data files will be stored in the runtime directory.

Configuration

Setting	Description
configuration.scheme	Specifies whether the REST based configuration API should be exposed via `http` or `https`.
configuration.schemeCertificate	Specifies the certificate used for SSL. **Note**: *This setting is applied when configuration.scheme is set to https.

Setting	Description
configuration.port	Specifies which http/https `port` the REST based configuration API listens on.
configuration.rootDirectory	Specifies the location for the product configuration component files for self-hosting. If this setting is not specified it defaults to a relative path used in the packaged installer (i.e., `../configuration`). If the setting is invalid or the path is invalid, the runtime will not self-host the configuration component.
configuration.centralConfig	Specifies if the hub is in central configuration mode, allowing it to be used to remotely configure other hubs. Set to True to enable this capability. The default setting is False.
configuration.autoSaveInterval	Specifies the maximum interval in seconds that the configuration file is saved after a configuration change is made. Defaults to 60 seconds. The minimum is 5 seconds and the maximum is 3600 seconds (1 hour).
configuration.backupCopies	Specifies the maximum number of configuration copies to backup. The default is 50. The minimum value is 0 (no backups) and the maximum is 100. When enabled, a 'backups' directory is created in the application directory. The existing configuration file is copied to the backup directory prior to saving any new changes to the configuration. If the maximum backups already exist, the oldest backup (based on the last time the file was modified) is deleted from the backup directory.

Logging

Setting	Description
log.fileSizeMB	Specifies the maximum event log file size in megabytes. The valid range is 10–1000. If this setting is not specified or is invalid, a value of 100 will be used. Once the file reaches the limit, the oldest events are moved into a backup file (.bak) and a new event log file is created. If a backup file already exists, it will be overwritten.
log.logAuditEvents	Specifies if the hub logs audit events for objects (Connections, Models, etc) that are changed. When enabled, audit events are logged for creation, update, and delete events. These events are logged a separate audit log database. Defaults to False. (When audit logging is changed from Enabled to Disabled a final Audit log event will be logged reporting that the setting was disabled)
log.debugLogging	Enables advance debugging. Options are NONE, ERROR, WARNING, INFO, DEBUG, and ALL. Only enable this setting in cases where detailed debug logs are required to troubleshoot an issue. Enabling this will impact performance. See Debug Logging for additional details.

Redundancy

Setting	Description
redundancy.backup.enabled	Specifies if the hub is running as a redundancy backup to another primary hub. See Application-Level Redundancy for more details.
redundancy.backup.primary.uri	Specifies the URI for the primary hub's configuration interface. This will be based on the primary's `configuration.scheme` and

Setting	Description
	`configuration.port` settings described above and its `host address`. For example, `http://127.0.0.1:45245`.
redundancy.backup.primary.pingIntervalSeconds	Specifies how often to ping the primary (in seconds) to ensure it is operational. The valid range is 1-3600. If this setting is not specified, a value of 10 will be used.
redundancy.backup.primary.pingAtt	Specifies how many failed attempts should occur before promoting the backup to active. The valid range is 1-10. If this setting is not specified, a value of 1 will be used.

Expression Engine (JavaScript)

Setting	Description
expressionEngine.allowExpressionImports	Specifies if the hub will allow the use of 3^rd^-party javascript imports in instance, condition, or trigger expressions. See Expressions for more details.
expressionEngine.allowExpressionEval	Specifies whether or not the use of eval() is allowed during javascript evaluation. This setting applies to all expression fields and imported libraries
expressionEngine.expressionEngineTimeoutSeconds	Specifies the maximum amount of time an expressions can take to compute a result. The valid range is 1-30 seconds. If this settings is not specified, a value of 10 seconds is used. If this setting is out-of-range the settings will be set to the closest acceptable value.
expressionEngine.hubBindingName	Specifies the name of the internal JavaScript object used in the expression engine to provide access to metadata like Quality. This only needs to be changed if importing a JS library that may have a naming conflict, which should be rare.
expressionEngine.allowEnvironmentVariables	Specifies if the hub can resolve environment variable references. If disabled `{{System.Environment}}` references will not be resolved and will not be available from the reference panel. The default setting is False. See Environment Variables for more information.

MQTT Broker

Setting	Description
hbMqttBroker.enabled	Enables the Intelligence Hub MQTT Broker.
hbMqttBroker.allowAnonymous	When false MQTT clients are required to specify a username and password to authenticate with the broker. In order for a user to access the broker the following claims are required: • `broker:subscribe`: Allows the MQTT client subscribe access to all topics. • `broker:publish`: Allows the MQTT client publish access to all topics. • `broker:*`: Allows the MQTT client subscribe and publish access to all topics. The default administrator user is not supported. See Create a User for details on creating a user with broker access.

Setting	Description
hbMqttBroker.endpoints	An array of one or more endpoints hosted by the broker. The endpoints have the following settings.
hostname	The hostname the endpoint binds to. By default this is 0.0.0.0, which binds to all endpoints.
port	The port the endpoint binds to. By default this is 1885.
useSSL	Enables secure communication (SSL) for the endpoint.
certificateKeyAlias	When SSL is enabled, this specifies the alias name for the key in the keystore to use for encryption. The public key must be trusted by MQTT Client.
protocol	The protocol used by the endpoint. The following options are available. • `mqtt.tcp` – TCP connection • `mqtt.ws` – Websocket connection

REST Data Server

Setting	Description
restDataServer.enabled	Enables the REST Data Server.
restDataServer.hostname	The hostname/network interface to bind to. In most cases this should be 0.0.0.0, 127.0.0.1 (localhost), or a NIC address on the machine
restDataServer.port	The port to bind to. Defaults to 8885.
restDataServer.useHttps	Enable this to host using https.
restDataServer.schemeCertificate	The certificate/key used to host https. This should be added to the certificate store. By default this uses the internal self-signed certificate.
restDataServer.mcpServerEnabled	Enables the MCP Server
restDataServer.mcpServerPort	The port to bind the MCP Server to. Defaults to 45345
restDataServer.mcpConfigTools	Enable MCP Tools for project configuration. Disabled by default. See MCP Services for details

After enabling the REST Data Server, access http://localhost:8885/data/doc to view and test the OpenAPI specification.

To use the MCP Server, the REST Data Server must be enabled. A valid Bearer Token is required for authentication. The MCP Server exposes all Pipelines configured with an API Trigger. Currently, only Server-Sent Events (SSE) are supported. For more information about MCP, see the official MCP documentation.

Security

Setting	Description
Session Timeout Interval	The amount of time an authenticated user session is valid for. This setting only applies to users that have logged in over the Configuration Server or REST Data Server API. Setting the timeout to 0 allows users to stay logged in until they manually log out or the application shuts down.
Authentication Preference	Determines whether users must re-authenticate when opening the UI in a new browser tab. If set to **Ask Every Time**, users are prompted to log in for each new tab. If set to **Stay Logged In**, users remain authenticated when opening additional tabs from an existing session. *NOTE: This setting only applies when there is an a configured session timeout.*

Git Backup

The Git Backup feature is used to automatically back-up IntelligenceHub configuration based on the configuration schedule. The backup is stored in a single JSON file and can be used by Git Deployment Integration.

Manual backups can be performed when the backup mode is set to **Manual** or **Scheduled**. During manual or scheduled backups changes to the hub configuration are pushed to the specified branch of the repository. When there are no changes since the previous backup, no commit is pushed. The commit message differentiates scheduled and manual backups. For manual backups, the IntelligenceHub user must have `Settings:execute_write` permission and the commit message will contain the user that performed the backup. See Users and Roles for more information.

Git Backup settings support the use of System and Environment references. Changes to System Variables will be detected automatically and cause the Git Backup schedule to be restarted. See System Variables and Environment Variables for more information.

Setting	Description
backup.mode	The mode controls how backups are performed. The following options are available. • `disabled` – No backups can be performed • `manual` – User-initiated backups only • `scheduled` – Scheduled (cron) and User-initiated backups.
backup.cronSchedule	The cron schedule to run the backup, no back-up is performed when blank. Default is daily at 3am UTC.
backup.branch	Branch in the repo to use for backup.
backup.deployFile	The relative path in the repository to save the configuration.
backup.repo.type	Git (git) is currently the only supported type.
backup.repo.uri	The URI of the Git repository.
backup.repo.author	Author to use in commit messages.
backup.repo.email	Email to use in commit messages.
backup.repo.auth.type	The authentication type. None (none) and Username/Password (pass) are currently supported.
backup.repo.auth.username	Git username to use for password authentication.
backup.repo.auth.password	Git password (or personal access token) to use for password authentication.

Example Template File with Defaults

```
{
  "settings": {
    "directories" : {
      "appData" : "../appData",
      "storeForwardData" : "../appData"
    },
    "configuration" : {
      "scheme" : "http",
      "port" : 45245,
      "autoSaveIntervalSeconds" : 60,
      "backupCopies" : 50,
      "centralConfig" : false
    },
```

```
    "hbMqttBroker" : {
      "enabled" : false,
      "allowAnonymous" : true,
      "endpoints" : [ {
        "hostname" : "0.0.0.0",
        "port" : 1885,
        "useSSL" : false,
        "certificateKeyAlias" : "",
        "protocol" : "mqtt.tcp"
      }, {
        "hostname" : "0.0.0.0",
        "port" : 1886,
        "useSSL" : false,
        "certificateKeyAlias" : "",
        "protocol" : "mqtt.ws"
      } ]
    },
    "restDataServer" : {
      "enabled" : false,
      "hostname" : "0.0.0.0",
      "port" : 8885,
      "useHttps" : false,
      "schemeCertificate" : ""
    },
    "log" : {
      "fileSizeMB" : 100,
      "logAuditEvents" : false
    },
    "redundancy" : {
      "backup" : {
        "enabled" : false,
        "primary" : {
          "uri" : "",
          "pingIntervalSeconds" : 10,
          "pingAttempts" : 1
        }
      }
    },
    "expressionEngine": {
      "allowExpressionImports": false,
      "expressionEngineTimeoutSeconds": 10,
      "hubBindingName": "hub",
      "allowEnvironmentVariables" : false
    }
  }
}
```

Command Line Options | User Guide

Source: https://guide.highbyte.com/setup/command_line/

Command Line Options

Intelligence Hub can be launched with several command line arguments. Below is a list of the available arguments along with their arguments.

Option	Arguments	Description
`start`	N/A	Start the Intelligence Hub runtime.
`stop`	N/A	Stop the Intelligence Hub runtime.
`version`	N/A	Print the Intelligence Hub version.
`export`	`<file_path>`	Export the deployment JSON to the specified file. If no file path is specified, the file will be exported as `intelligencehub-deployment.json`
`encrypt`	`<file_path>`	Encrypt secrets in the provided JSON file.
`set-password`	`<new_password>`	Set the password for the keystore.
`rotate-password`	`<current_password> <new_password>`	Rotate the password for the keystore.

Usage

start

Start the Intelligence Hub runtime using a classpath wild-card to support loading custom dependencies.

```
java -cp "intelligencehub-runtime.jar:lib/*" com.highbyte.intelligencehub.runtime.Main start
```

stop

Stops the Intelligence Hub runtime process.

```
java -jar intelligencehub-runtime.jar stop
```

version

Prints the Intelligence Hub version

```
java -jar intelligencehub-runtime.jar version
```

export

Exports the deployment file intended to be used for the Git Deployment Integration.

```
java -jar intelligencehub-runtime.jar export my_export.json
```

encrypt

Encrypts the specified JSON file in-place using IntelligenceHub password encryption key and rules.

```
java -jar intelligencehub-runtime.jar encrypt my_encrypted.json
```

set-password

A password for the Intelligence Hub Key store file can be set one of three ways:

Option 1: As clear text directly in the command line.

```
java -jar intelligencehub-runtime.jar set-password my_password
```

Option 2: As an environment variable, where the environment variable name is prefixed with `env:`.

```
java -jar intelligencehub-runtime.jar set-password env:MY_PASSWORD_ENV_VAR
```

Option 3: Interactively via the console. This is achieved by not passing a password as part of the command line.

```
java -jar intelligencehub-runtime.jar set-password
```

The `set-password` command can only be used once to change the password used to encrypt the certificate store file. After `set-password` is run, the environment variable `HIGHBYTE_STORE_PASSWORD` must be set. It should be equal to your new password.

To change the password, see `rotate-password`

rotate-password

A password for the Intelligence Hub Key Store can be rotated one of three ways:

Option 1: As clear text directly in the command line.

```
java -jar intelligencehub-runtime.jar rotate-password my_current_password my_new_password
```

Option 2: As environment variables, where the environment variable name is prefixed with `env:`.

```
java -jar intelligencehub-runtime.jar rotate-password env:MY_CURRENT_PASSWORD_ENV_VAR env:MY
```

Option 3: Interactively via the console. This is achieved by not passing the passwords as part of the command line.

```
java -jar intelligencehub-runtime.jar rotate-password
```

`rotate-password` can only be run after the default password is changed using `set-password`. After `rotate-password` is run, the environment variable `HIGHBYTE_STORE_PASSWORD` must be set. It should be equal to your new

password.

Docker | User Guide

> Source: https://guide.highbyte.com/setup/docker/

Docker

The Docker image file `HighByte-Intelligence-Hub-X.X.X Docker (Build XXXX.X.X.X).tar` includes the Intelligence Hub runtime and configuration components, and is built from the alpine linux image with OpenJDK 21. This section covers how to install and run the docker image.

This is an x86_64 image. Other CPU architectures such as arm64 require building your own image from the installer.

Install

1. Run the following docker command to load the docker image:

```
docker load -i "HighByte-Intelligence-Hub-X.X.X Docker (Build XXXX.X.X.X).tar"
```

2. Setup a volume to host the application configuration. This step is optional, but recommended so that you do not lose your configuration when performing future image updates.

```
docker volume create hb_vol
```

3. Run the image. The below command includes making all ports available, but you may choose to restrict the ports. Remove the –mount source=hb_vol if not using a volume.

```
docker run -p 45245:45245 -p 1885:1885 -p 8885:8885 -p 45345:45345 --name highbyte --mou
```

Below is a list and description of each of the default ports.

Port	Description
45245	REST API used for configuration.
1885	Internal MQTT Broker.
8885	REST data server.
45345	MCP server.

Changes to the ports require the container to be recreated. Changing the ports from the user interface will not change the container ports and may interrupt connections.

Git Integration | User Guide

Source: https://guide.highbyte.com/setup/git/

Git Integration

This page documents how the IntelligenceHub Integrates with git during start-up in order to fetch and layer hub configurations from git. To learn more about Git Backup, see: Git Backup.

All IntelligenceHub start-up Git integration is configured using a JSON file. This file is specified using the environment variable `HIGHBYTE_DEPLOYMENT_FILE` or java property `highbyte.deployment.file`. When set, the IntelligenceHub will fetch configuration from one or more Git repositories during start-up. Any errors that occur during configuration are considered critical and will prevent the product from starting.

The deployment settings file can be configured to pull one or many JSON files from Git/Disk, layer the content together, and form a single Deployment settings file that an IntelligenceHub instance is configured with during start-up.

Any secrets stored in the configuration are encrypted/decrypted using the instance json encryption key stored in its certificate store.

Deployment Settings JSON file Schema

```
/**
 * Represents the configuration object for a deployment settings.
 */
{
    /**
    * The version of the configuration object, currently 0
    */
    version: 0,

    /**
     * Defines the Git repositories used during deployment. Fragments reference a repository
     */
    repos?: {
        type: "git",                // The type of repository, currently only "git" is suppo
        name: string,               // The local name used in this file to link the fragment
        uri: string,                // The URI of the repository.
        author: string,             // The author of the repository.
        email: string,              // The email address of the repository author.
        /**
         * Authentication scheme
         */
        auth: {
            type: "pass" | "none",  // The type of authentication, either "pass" (password)
            password?: string,      // The password for authentication.
            username?: string,      // The username for authentication.
        }
    }[],

    /**
     * An array of fragment configurations.
```

```
     */
    fragments: {
        details: {
            type: "git" | "file",    // The type of fragment, either "git" or "file".
            deployFile: string,      // The path to the deployment file.
            repoName?: string,       // Git only. The name of the repository associated with
            ref?: string,            // Git only. The reference (e.g., branch or tag) in the
        },
        keyPaths: string[]           // JSON Pointer-like string to select a sub-set of eleme
    }[]
}
```

Passwords or PATs in the deployment settings file can be encrypted using the application cli:

```
java -jar intelligencehub-runtime.jar encrypt </path/to/settings.json>
```

The source-controlled JSON deployment file defines all project and instance configuration used by an IntelligenceHub instance. This format is also used by the Git Backup features configured in Application Settings and can be exported using application cli:

```
java -jar intelligencehub-runtime.jar export [path]
```

Deployment File JSON High-Level Schema

```
/**
 * Describes the high-level elements of the deployment JSON file.
 */
{
    /**
     * Information about the instance that generated the deployment file.
     */
    productInfo: {
        company: string,
        product: string,
        version: string,
        build: string,
        stage: string,
    },

    /**
     * Licensing information.
     */
    license: {
        signature: string,
        license: string,
    },

    /**
     * Project configuration.
     */
    project: {
        version: number,
        connections: object[],
        conditions: object[],
```

```
        inputs: object[],
        outputs: object[],
        modeling: {
            models: object[],
            instances: object[],
        },
        pipelines: object[],
        tags: object[],
        namespaces: object[],
        functions: object[],
    },

    /**
     * Network-related configuration.
     */
    network: {
        version: number,
        groups: object[],
        hubs: object[],
    },

    /**
     * Access control configuration.
     */
    accesscontrol: {
        version: number,
        roles: object[],
        user: object[],
        apiKeys: object[],
    },

    /**
     * System-level configuration.
     */
    system: {
        version: number,
        secrets: object[],
        variables: object[],
        certificates: object[],
        settings: object,
    },
}
```

Git Example | User Guide

Source: https://guide.highbyte.com/setup/git/example/

Git Example

All secrets in the JSON configuration files are encrypted using an AES key stored in the intelligencehub-certificates.pkcs12 file. To deploy the configuration on another Intelligence Hub instance, this certificate file must be copied to the instance's /appData directory before deployment.

The following example shows how to use the IntelligenceHub deployment settings to:

- Pull a model and instance configuration using Git
- Combine the configuration with a connection and input stored locally, and
- Start the IntelligenceHub with the combined configuration.

1. Setup the deployment settings JSON file

The deployment settings file below defines two fragments:

- Fragment 1: Uses the repo defined in the file by setting `"repoName": "highbyte"` and gets deployment JSON from a file named `model-deployment.json` at the root of the repository. The keyPath selects only the modeling components of this file.
- Fragment 2: References a deployment file stored at `/usr/local/highbyte/deployments/connection-deployment.json` and uses keyPaths to select only connections, inputs, and outputs from this file.

```
{
  "version" : 0,
  "repos" : [ {
    "type" : "git",
    "uri" : "<uri>",
    "name" : "highbyte",
    "author" : "",
    "email" : "",
    "auth" : {
      "type" : "pass",
      "password" : "<token>",
      "username" : "<user>"
    }
  } ],
  "fragments" : [ {
    "details" : {
      "type" : "git",
      "deployFile" : "model-deployment.json",
      "repoName" : "highbyte",
      "ref" : "models"
    },
    "keyPaths" : [ ".project.modeling" ]
  }, {
    "details" : {
      "type" : "file",
      "deployFile" : "/usr/local/highbyte/deployments/connection-deployment.json"
```

```
    },
    "keyPaths" : [ ".project.connections", ".project.inputs", ".project.outputs" ]
  } ]
}
```

To encrypt the token in the deployment settings file (or any highbyte JSON file), use the `encrypt` command. For example:

```
java -jar intelligencehub-runtime.jar encrypt deployment-settings.json
```

This command encrypts `deployment-settings.json` in place, the result is shown below, note the password has been encrypted.

```
{
  "version" : 0,
  "repos" : [ {
    "type" : "git",
    "uri" : "<uri>",
    "name" : "highbyte",
    "author" : "",
    "email" : "",
    "auth" : {
      "type" : "pass",
      "password" : {
        "type" : "Encrypted",
        "value" : {
          "keyId" : "AnBU7AXftXCS7+EZNM+9Zw==",
          "iv" : "gEma6X9PPh38XoqC1dFcew==",
          "ciphertext" : "dQMlw3rRfzBVrVxfW5a/LQ=="
        }
      },
      "username" : "<user>"
    }
  } ],
  "fragments" : [ {
    "details" : {
      "type" : "git",
      "deployFile" : "model-deployment.json",
      "repoName" : "highbyte",
      "ref" : "models"
    },
    "keyPaths" : [ ".project.modeling" ]
  }, {
    "details" : {
      "type" : "dir",
      "deployFile" : "/usr/local/highbyte/deployments/connection-deployment.json"
    },
    "keyPaths" : [ ".project.connections", ".project.inputs", ".project.outputs" ]
  } ]
}
```

2. Setup deployment JSON files

Deployment files containing an IntelligenceHub instance configuration can be exported using the `export` command. The following example exports a deployment json to `/usr/local/highbyte/deployments/connection-`

`deployment.json`. The `export` command exports to `intelligencehub-deployment.json in the working directory when a path is not specified.

```
java -jar intelligencehub-runtime.jar export /usr/local/highbyte/deployments/connection-depl
```

This example deployment settings combines the model-deployment.json stored in the `model` branch of the configured repo with the connection-deployment.json stored in `/usr/local/highbyte/data/intelligencehub-deployment.json`.

3. Start the IntelligenceHub using the deployment JSON files

Specify the deployment settings file using an environment variable or java property.

Example Linux start script:

```
HIGHBYTE_DEPLOYMENT_FILE=./deployment-settings.json ./start-linux.sh
```

Example using java command on windows:

```
java !JAVA_OPTS! -Dhighbyte.deployment.file="./deployment-settings.json" -jar intelligencehu
```

The IntelligenceHub uses the settings json file to reconfigure the hub at start-up if either the environment variable or java property are set. Any errors the occur during this process will stop start-up, error information is printed to console stdout.

An example of the resulting hub deployment configuration is shown in combined-deployment.json.

High Availability | User Guide

Source: https://guide.highbyte.com/setup/high_availability/

High Availability

Overview

High Availability (HA) mode allows the Intelligence Hub to run across multiple nodes, ensuring continuous operation and failover. Only one node is active at a time; standby nodes are ready to take over if the primary node fails. This is managed through heartbeats and file synchronization.

Heartbeats and File Synchronization

- **Heartbeats:** Each node periodically signals its status. If the primary node stops sending heartbeats, a standby node automatically takes over.
- **File Synchronization:** Configuration files, Extension cache, and Pipeline State are regularly synced to ensure a standby node can seamlessly assume the primary role.

Setup

To add a new node, copy the `intelligencehub-certificatestore.pksc12` file to the node's application data directory.

This initial step of copying the certificate store file allows the secondary hub to store secrets using the same encryption schema as the primary hub. This ensures both hubs can securely exchange and access encrypted data like secrets, passwords, or certificate keys. For example, when a new connections requiring a system secret is created in the primary hub, both the connection and system secret will be securely synchronized to the secondary hub.

Finally, assign a unique node ID to each new node.

Command Line Usage

Start the HA runner using the command line. The main commands are:

- `create` : Initializes the failover service, syncs initial configuration and state files, and sets the node ID as the preferred primary node.
  - The preferred primary node is always selected as primary when available.
- `start` : Starts a node in HA mode, joining the failover group.
- `help` : Displays usage information.

Example Usage

```
java -cp "intelligencehub-runtime.jar;lib/*" com.highbyte.intelligencehub.runtime.HAMain [op
```

Option	Description	Example Value
`-j` , `--jdbcURI`	JDBC URI for the node (can use environment variables)	`jdbc:postgresql://localhost:5432/dbName?user=username&password=password` `env:URI`

Option	Description	Example Value
`-n` , `--nodeId`	Unique node ID for the HA instance	

You can use environment variables for arguments by prefixing with `env:` (e.g., `-j env:URI`).

Example Commands

```
java -cp "intelligencehub-runtime.jar;lib/*" com.highbyte.intelligencehub.runtime.HAMain cre
java -cp "intelligencehub-runtime.jar;lib/*" com.highbyte.intelligencehub.runtime.HAMain sta
```

Agents

Agents provide the interface for communication between the Intelligence Hub node and the High Availability environment. Currently, only the PostgreSQL agent is supported.

Help

For help, run:

```
java -cp "intelligencehub-runtime.jar;lib/*" com.highbyte.intelligencehub.runtime.HAMain hel
```

Installation | User Guide

> Source: https://guide.highbyte.com/setup/installation/

Installation

Before starting the installation, ensure all [required dependencies] are installed.

Extract Installation

1. Extract `HighByte-Intelligence-Hub-x.x` to a location where you have read/write/execute privileges in order run HighByte Intelligence Hub.

Setup Application Data Directory (Recommended)

By default, the application configuration and settings are stored in the `appData` directory within the product zip. The application start scripts use the `appData/intelligencehub-settings.json` file to locate application state and store-forward data in this location. It is recommended that appData be maintained this way to make product upgrades easy. The `appData` and `storeForward` directories are controlled using the the `settings.directory` object in the `intelligencehub-settings.json` file.

Example: Store application data in a custom location

Assume a user would prefer to store appData and storeForwardData in a custom location of their home directory. Before launching the runtime, open `runtime\appData\intelligencehub-settings.json` in a text editor. Update the appData and storeForwardData settings as shown, make sure the custom directories exist, and configure permissions to allow the application to read/write to these directories. The next time the application is started all application and store-forward data is located into the custom directories. Data stored previously can not be automatically relocated.

```
"settings": {
    "directories" : {
        "appData" : "C:\\HighByte\\appData",
        "storeForwardData" : "C:\\HighByte\\storeForwardData"
    },
}
```

Note on Windows backslashes must be escaped using `\\` syntax.

Start Runtime

To launch the runtime in a command shell, open a shell and navigate to the runtime directory. Start the runtime by executing the `start-[OS]` file associated with your OS (e.g., `start-windows.bat`, `./start-linux.sh`, `./start-macos.sh`).

The runtime will begin logging event messages to the console upon startup.

Stop Runtime

To stop the runtime safely, execute the corresponding `stop-[OS]` script. This will ensure the runtime completes any outstanding work (e.g., persisting configuration updates, storing data for future forwarding, etc.) before shutting down.

Run as Windows Service

When running on Windows it is recommended to run as a service. To install the Intelligence Hub as a service run the `runtime\service\install-windows-service.bat` script. Note, this feature requires Java to be in the system PATH environment variable. If you are unsure if this was set when you installed Java check out this link: https://www.java.com/en/download/help/path.html.

Once installed, the service can be found in Windows Services Manager under the name HighByte Intelligence Hub Runtime.

Executing the file `runtime\service\uninstall-windows-service.bat` will stop and uninstall the service.

Setup system.d service on Linux

NOTE: In order to install to the example directory and use `systemctl` you will need super user permission.

Copy the `runtime`, `configuration`, and `appData` directories from `HighByte-Intelligence-Hub-x.x` to the desired installation directory. This example will use `/opt/highbyte`.

Once complete, the directory listing should look like:

```
$ ls /opt/highbyte/
configuration  runtime appData
```

Make sure the start and stop scripts have execute permissions: `sudo chmod +x /opt/highbyte/runtime/*-linux.sh`

Create `/etc/systemd/system/highbyte.service`. This minimal example service file will automatically start the Intelligence Hub during system boot-up.

To install the service run: `sudo systemctl enable highbyte.service`

To start the service run: `sudo systemctl start highbyte.service`

To verify the service started successfully run: `sudo systemctl status highbyte.service`

Application State | User Guide

> Source: https://guide.highbyte.com/setup/installation/application_state/

Application State

The Intelligence Hub installation includes the application code, configuration, and application settings. This section describes the different files in the installation.

Application Code

The following files and directories are the core application code of the Intelligence Hub. This code changes between releases.

- `configuration/*`
  - The web based user interface that is accessible via web browser.
- `runtime/intelligencehub-runtime.jar`
  - Core runtime Java code.
- `runtime/lib/*`
  - All 3rd party Java dependencies.

Application Data

The following files and directories are included with the installation.

- `appData/*`
  - Directory containing file templates and a default `intelligencehub-settings.json` configured to use `appData` for all application state and store-forward data.

Configuration State

The following files are part of the Intelligence Hub configuration state.

- `appData/intelligencehub-configuration.json`
  - The project configuration for the Intelligence Hub (e.g., connections, models, pipelines, etc.) stored as JSON.
- `appData/intelligencehub-certificatestore.pkcs12`
  - Certificate store that contains the keys for encrypting passwords in the `intelligencehub-configuration.json` file.
- `appData/intelligencehub-users.json`
  - Contains local users, password hashes, roles and claims settings.
- `appData/intelligencehub-events.db`
  - Event log files stored in an encrypted SQLite format.
- `appData/intelligencehub-events.log`
  - Event log files stored in a JSON based text format.

- `appData/intelligencehub-cache.db`
    - Cached state of connections in the Intelligence Hub (e.g., SQL indexed query, getting only data newer than a certain date, etc.).
- `appData/intelligencehub-writestore-CONECTIONNAME.db`
    - Store and forward file for a specific connection (only present when store and forward is enabled on the connection).
- `appData/backups/*`
    - A folder containing automatic backups of the `intelligencehub-configuration.json`. Snapshots are added anytime the configuration is modified.

Application State

The following files hold application state independent of the configuration.

- `appData/intelligencehub-settings.json`
    - Contains application settings like the REST API port, location of configuration directory, max events to log, etc. See Application Settings for details.
- `appData/intelligencehub-remoteconfig.json`
    - Contains the URL of the central Intelligence Hub instance to connect to and the token required to authenticate the connection. This is only applicable when using central configuration. See Remote Hub Configuration for details.
- `appData/intelligencehub-secrets.json`
    - Contains encrypted secrets bound to this installation instance. Secrets can be referenced by a project to enable installation-specific credentials.
- `appData/intelligencehub-systemvariables.json`
    - Contains variables bound to this installation instance. System Variables can be referenced by a project to enable installation-specific connection settings that don't need to be secured such as hostname(s) or port(s).

Installing Java | User Guide

> Source: https://guide.highbyte.com/setup/installation/java/

Installing Java

Java Runtime Environment is required to run HighByte Intelligence Hub.

Check to see if a JRE is already installed

1. From a command or terminal prompt execute `java -version`
2. If the command succeeds and the java version is 21 or greater there is no need to continue. Otherwise, proceed to the next section.

Install OpenJDK JRE

1. Download OpenJDK 21 or greater at OpenJDK
2. Follow the instructions that accompany the OS dependent distribution that was downloaded to install the JRE for your system.
3. Close any command or terminal prompt windows that are currently open.
4. Re-run Step 1 to verify the JRE was properly installed.

Licensing | User Guide

> Source: https://guide.highbyte.com/setup/installation/licensing/

Licensing

In unlicensed mode the Intelligence Hub runs for two hours at a time, fully functional. After the two hours expires, all users are logged out and all runtime services are stopped, with the exception of configuration services. To reset the trial the user must re-login and reset the runtime to run for another two hours. To run for longer periods a valid license must be installed.

To license the Intelligence Hub, follow the instructions below.

Download a License

1. Download your license from the licensing portal.

Installing the License through the UI

1. Within HighByte Intelligence Hub, navigate to Settings and then License.
2. Click the “Choose File” button at the bottom of the page and browse to the license file downloaded from the portal.
3. Click the “Import License” button.
4. Once licensed the trial banner will no longer be visible and the License Status will update to the new license.

Installing the License Manually

1. Create a `license` directory in the appData directory. By default, this would be `appData\license`.
2. Copy the license file to this directory and name the file `intelligencehub.license`.
3. Restart the runtime.

OpenTelemetry | User Guide

> Source: https://guide.highbyte.com/setup/open_telemetry/

OpenTelemetry

The start scripts demonstrate how the Intelligence Hub can be used with the OpenTelemetry Java Agent. Modification may be required for deploying in a Production environment.

Using OpenTelemetry with HighByte Intelligence Hub

The HighByte Intelligence Hub start scripts support running with the OpenTelemetry Java Agent out-of-the-box. Intelligence Hub metrics, logs, and JVM exporters are currently supported.

When starting with the OpenTelemetry Java Agent without the optional `endpoint` argument, OpenTelemetry metrics and logs are sent to `localhost:4318`.

Windows

```
.\start-windows.bat otel [otel endpoint]
```

Linux

```
./start-linux.sh otel [otel endpoint]
```

macOS

```
./start-macos.sh otel [otel endpoint]
```

Docker

The docker entrypoint supports the `otel` command and the collector endpoint is required because there is no collector running on the container. See Docker for more information.

```
docker run -p 45245:45245 -p 1885:1885 -p 8885:8885 -p 45345:45345 --name highbyte --mount s
```

Application Logs

All INFO, WARN, and ERROR event log messages are exported to OpenTelemetry. Logs are exported with the following attributes:

Attribute Name	Description
highbyte.hub.eventlog.source	Name of the component generating the event log message.
highbyte.hub.eventlog.type	Corresponding to the HighByte message type; INFO, WARN, and ERROR

Connection Metrics

The following metrics are exported for monitoring connections in HighByte Intelligence Hub:

Read Metrics

Metric Name	Type	Description	Attributes
`highbyte.hub.connection.stats.read.pending.size`	Histogram	Number of pending reads for a connection	`highbyte.hub.connection.name`

Metric Name	Type	Description	Attributes
		across all inputs	
`highbyte.hub.connection.stats.read.count`	Counter	Total number of reads executed for a connection's input	`highbyte.hub.connection.name` `highbyte.hub.connection.input.n`
`highbyte.hub.connection.stats.read.error.count`	Counter	Total number of read errors for a connection's input	`highbyte.hub.connection.name` `highbyte.hub.connection.input.n`
`highbyte.hub.connection.stats.read.duration`	Histogram	Read duration for a connection's input in seconds	`highbyte.hub.connection.name` `highbyte.hub.connection.input.n`

Write Metrics

Metric Name	Type	Description	Attributes
`highbyte.hub.connection.stats.write.pending.size`	Histogram	Number of pending writes for a connection across all outputs	`highbyte.hub.connection.name`
`highbyte.hub.connection.stats.write.dropped.count`	Counter	Number of dropped writes for a connection across all outputs	`highbyte.hub.connection.name`
`highbyte.hub.connection.stats.write.count`	Counter	Total number of writes for a connection's output	`highbyte.hub.connection.name` `highbyte.hub.connection.outpu`
`highbyte.hub.connection.stats.write.error.count`	Counter	Total number of write errors for a connection's output	`highbyte.hub.connection.name` `highbyte.hub.connection.outpu`
`highbyte.hub.connection.stats.write.duration`	Histogram	Write duration for a	`highbyte.hub.connection.name` `highbyte.hub.connection.outpu`

Metric Name	Type	Description	Attributes
		connection's output in seconds	

Pipeline Metrics

The following metrics are exported for monitoring pipelines in HighByte Intelligence Hub:

Trigger Metrics

Metric Name	Type	Description	Attributes
`highbyte.hub.pipeline.stats.trigger.count`	Counter	Total number of triggers for a pipeline's trigger	`pipeline.name` `pipeline.trigger.name`
`highbyte.hub.pipeline.stats.trigger.error.count`	Counter	Total number of errors for a pipeline's trigger	`pipeline.name` `pipeline.trigger.name`

Execution Metrics

Metric Name	Type	Description	Attributes
`highbyte.hub.pipeline.stats.run.count`	Counter	Total number of runs for a pipeline	`pipeline.name`
`highbyte.hub.pipeline.stats.write.pending.size`	Histogram	Number of queued writes for a pipeline	`pipeline.name`
`highbyte.hub.pipeline.stats.wait.duration`	Histogram	Wait duration for a pipeline in seconds	`pipeline.name`
`highbyte.hub.pipeline.stats.execution.duration`	Histogram	Execution duration for a pipeline in seconds	`pipeline.name`
`highbyte.hub.pipeline.stats.error.count`	Counter	Total number of errors for a pipeline's stage	`pipeline.name` `pipeline.stage.name`

See JVM Metrics for more information about JVM metrics generated by the Open Telemetry Java Agent.

Redundancy | User Guide

> Source: https://guide.highbyte.com/setup/redundancy/

Redundancy

Application-level redundancy allows a second instance of the Intelligence Hub to act as a backup in the event the primary instance fails.

Set up application redundancy

1. Install two copies of the HighByte Intelligence Hub. Identify one as the primary instance and the other as the backup instance.
2. Configure and start the primary instance.
3. Configure and start the backup instance.
    - Enable the backup instance by updating the redundancy settings either through the UI Settings page or by editing the `intelligencehub-settings.json` manually. See Application Settings for details.

With the primary and backup instances running, application-level redundancy operates as follows:

1. The backup starts in standby mode. No pipelines are started.
2. The backup pings the primary's `v1/info` endpoint at a rate set by `pingIntervalSeconds`. In the event the primary does not respond, the backup continues to ping the primary until the number of successive failures is equal to `pingAttempts`.
3. Once failed `pingAttempts` is reached, the backup transitions to active mode and any pipelines that are enabled are started.
4. In active mode, the backup continues to ping the primary at a rate set by `pingIntervalSeconds`.
5. Once the backup successfully pings the primary, all pipelines stop and the backup transitions to standby mode.
6. The process repeats until the backup hub is stopped.

There is no automatic project synchronization between the primary and backup instances. The user needs to set up or deploy (through central hub management) a project for the primary and backup.

System Requirements | User Guide

Source: https://guide.highbyte.com/setup/system_requirements/

System Requirements

For the best experience we recommend the below requirements are met.

Hardware

The following system hardware requirements must be met to run HighByte Intelligence Hub:

- 1.4 GHz Processor
- 1 GB RAM
- 1 GB Available Disk Space
- Network Capable (TCP)

Note: These are minimal system requirements. Actual hardware requirements will vary based on product configuration.

Operating System

- Windows Server 2012 / 2016 / 2019 / 2022
- Windows 8 / 10 / 11
- Linux
- MacOS

Browser

The Intelligence Hub is configured using a browser, and by default is accessible on http://localhost:45245. The following browser versions are fully supported.

- Chrome >= 87
- Edge >= 88
- Firefox >= 78
- Safari/iOS >= 14
- Opera >= 73
- ChromeAndroid >= 125

JVM

The Intelligence Hub runtime requires between Java SE 21 or greater. HighByte recommends using OpenJDK.

Upgrade | User Guide

> Source: https://guide.highbyte.com/setup/upgrade/

Upgrade

Follow the steps below to perform an upgrade.

See Upgrading from 3.4 to 4.0 for more details upgrading from 3.x to 4.x.

1. Stop HighByte Intelligence Hub if it is currently running
   - Running as a Windows Service: use `uninstallwindows-service.bat` command.
   - Running on Linux, Mac or on Windows as an app: use `stop-[OS]` command.
2. Backup the application data and store and forward directories in a zip. This is best practice in the event the upgrade fails, or you decide to downgrade.
3. Install a fresh copy of the latest version to a new folder (e.g., `/HighByte/IntelligenceHub-x.x`).
4. Copy the appData from the old installation into the new installations appData directory.
5. Launch the new installation. This will perform any upgrades required by the latest version of the product.

Downgrade

To downgrade to an older version perform the following steps. Note: It is important to backup the application data and store and forward files before each upgrade, as files may be updated and no longer compatible with older versions.

1. Unzip or backup the original application data and store and forward directories.
2. Run the older version, and make sure appData and store and forward directories set in the `intelligencehub-settings.json` point to the original directories.

Upgrading from 3.4 to 4.0 | User Guide

Source: https://guide.highbyte.com/setup/upgrade/34_to_40_upgrade/

Upgrading from 3.4 to 4.0

This section covers upgrading an existing 3.X installation to 4.0, including any manual migration that is required. In most cases, no manual migration is necessary.

Upgrading the JVM

Version 4.0 requires minimum a JVM of 21. If you were not using v21 for 3.X you will need to upgrade the JVM.

Upgrading Configuration and State to 4.0

4.0 by default places all application state in an appData directory, next to the existing runtime and configuration directories.

If the old installation relied on the default behavior of storing application state in the runtime directory or relied on the default behavior of finding the intelligencehub-settings.json in the runtime directory, you will need to manually move application state files before launching 4.0. Follow the steps below.

1. Shutdown the existing 3.4 (or older) runtime
2. Backup the existing 3.4 installation to ensure no data is lost
3. Unzip the new 4.0 installation in a new directory
4. Copy the following files from the 3.4 runtime directory to the 4.0 appData directory
    - intelligencehub-settings.json
        - If the settings.directories.appData setting was used in this file to store application state in another directory then only move this file and ignore the rest
    - Any and all files with a .json extension
        - intelligencehub-configuration.json
        - intelligencehub-remoteconfig.json
        - intelligencehub-secrets.json
        - intelligencehub-systemvariables.json
        - intelligencehub-users.json
    - Any and all files with a .db extension
    - intelligencehub-certificatestore.pcks12
    - intelligencehub-state.dat
    - intelligencehub-events.json
    - intelligencehub-events.log
5. Run the new 4.0 installation. 4.0 will find the appData files, upgrade them as needed and launch the Intelligence Hub.

Upgrading Remote Configuration

If your old installation was using remote configuration, you'll need to upgrade all Intelligence Hub instances to version 4.0. This includes central and remote hubs. The protocol for remote configuration was updated significantly to support future migrations and new capabilities.

Other

{{System.Internal.Datetime}} references

The reference {{System.Internal.Datetime}} and other Datetimes are now bound to JavaScript as a Date data type. JavaScript expressions like "{{System.Internal.Datetime}} + 1" will now evaluate to a string in the form "Tue Jun 11 2024 18:50:00 GMT+0000(GMT)1". In 3.x this would evaluate to a number (unix epoch time + 1). To keep the previous behavior, update the expression as follows "{{System.Internal.Datetime}}.getTime() + 1".

{{System.Flows}} and {{System.Flows.Flowname}} references

In 4.0 all existing Flows are migrated to pipelines. This means if system references were being used to monitor flow status, these must be manually upgraded to {{System.Pipelines}} or {{System.Pipeline.PipelineName}} references. Pipelines have more advanced statistics to perform better troubleshooting.

Removed Global REST Data API Tokens

If the old installation was using a global API token, configured under Settings, these will no longer work in 4.0. You must create a new API Token that is linked to a User Account. These new tokens can be global, but can also be scoped to limit access.

v1/ REST API Configuration Routes

The v1/model and v1/instance routes have been removed. If these were being used to build models/instances we recommend instead using the v1/project/import and export APIs that can be found in the guide.

hub.withMetaData{{Connection.opc.tag}}

This method only works with Instances in Legacy Mode to support upgrading. This method is not available for Instances with Legacy Mode disabled. As an alternative, enable the Include Metadata option for the OPC input to get the timestamp and quality for the tag.

Event.metadata | User Guide

Source: https://guide.highbyte.com/tags/event.metadata/

Event.metadata

Event.value | User Guide

Source: https://guide.highbyte.com/tags/event.value/

Event.value

Troubleshooting | User Guide

> Source: https://guide.highbyte.com/troubleshooting/

Troubleshooting

HighByte Intelligence Hub includes several tools to help identify and fix errors that may occur during operation. These tools include the event log, status (connections and pipelines), and pipeline metrics. This section explains each and how best to use them.

- Event Log
- Status & Metrics
- Dashboard
- Debug Logging
- Debugging Pipelines

Dashboard | User Guide

> Source: https://guide.highbyte.com/troubleshooting/dashboard/

Dashboard

The dashboard is the main landing page for the Intelligence Hub. It provides a high level overview of the following:

- Current connection and pipeline statuses
- Intelligence Hub version, last restart time, and connected agent versions
- Key configuration settings
- Recent logs
- Time series charts of error logging
- Aggregate counts of conditions, models, instances, namespaces, connections, and pipelines

Dashboard Widgets

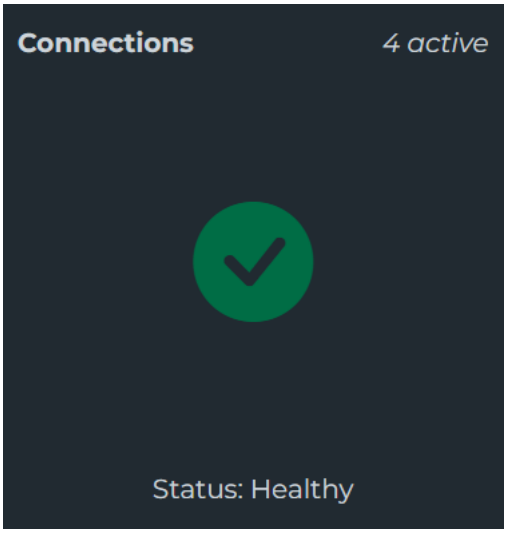
Hub

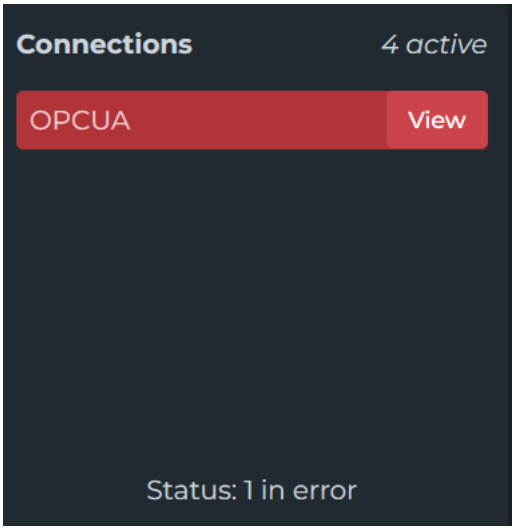
The Hub widget shows the current Intelligence Hub version, the last restart time, and versions of any connected agents (e.g., PI Agents). The Hub widget can be useful to quickly spot and update outdated agent versions.

Connections

The Connections widget provides a synopsis of all *active* connections in the Intelligence Hub, along with any active errors that need to be resolved.

If a connection is in error, click anywhere on its alert to go directly to the connection details. When no active connections are in error, the widget displays a healthy indicator.





Pipelines

The Pipelines widget provides a synopsis of all *active* pipelines in the Intelligence Hub, along with any active errors that need to be resolved or dismissed.

If a pipeline is in error, click anywhere on its alert to go directly to the pipeline details. When all pipelines run without errors, the widget displays a healthy indicator.

Overview

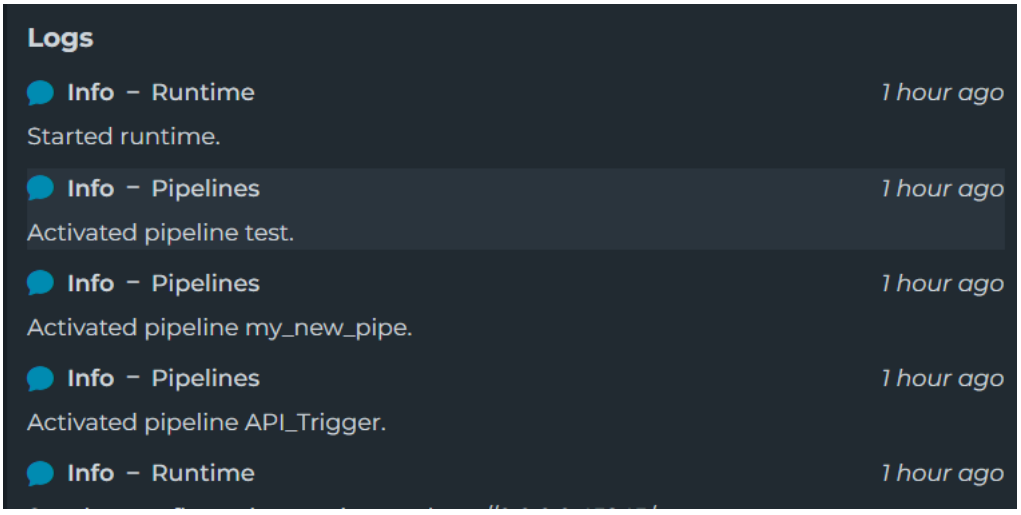
The Overview widget displays aggregate counts of all configured connections, conditions, models, instances, namespaces, and pipelines (active and inactive).

Click any row to navigate to the respective page to add, remove, or edit that item type.

Logs

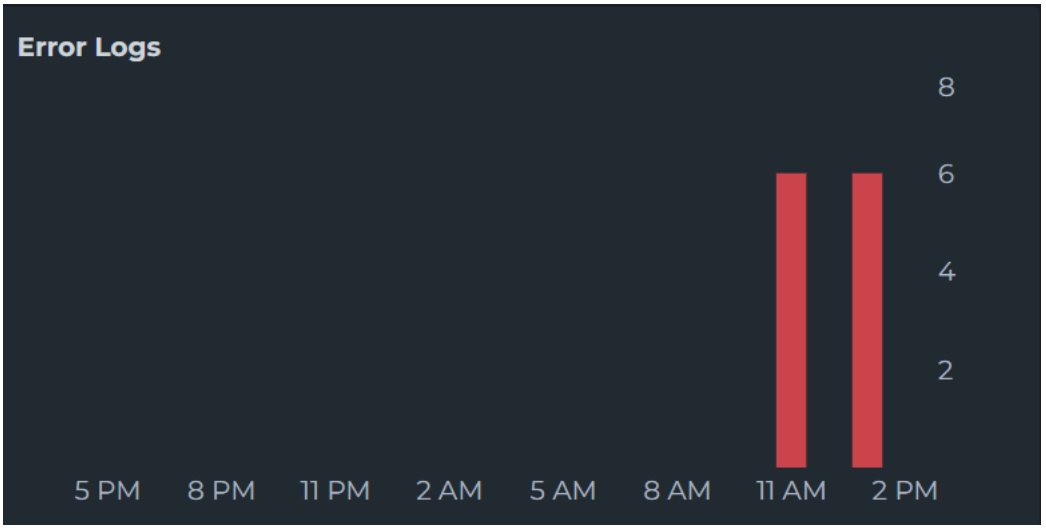
The Logs widget lists the most recent Intelligence Hub log entries including level (error, warning, info), relative timestamp, source, and message.

Click a log row to navigate to the Log Events page, filtered to that log timestamp and preceding log entries.



Error Logs Chart

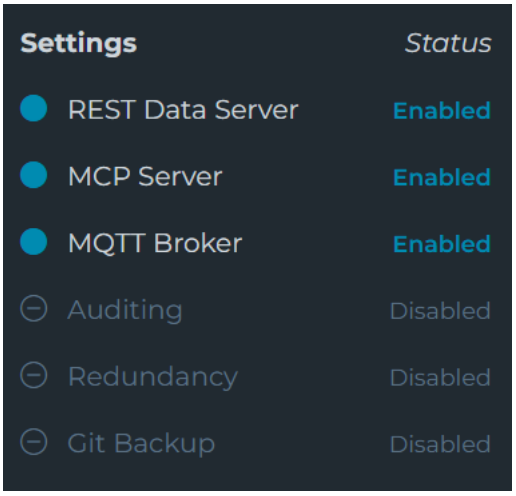
The Error Logs widget charts error counts for the past 24 hours. Hover over a bar to see the total for that hour. Click a bar to navigate to the Log Events page, filtered to the selected hour.



Settings

The Settings widget shows whether each listed Hub setting is enabled or disabled.

Click a row to navigate directly to that setting on the Settings page.



Debug Logging | User Guide

> Source: https://guide.highbyte.com/troubleshooting/debug_logging/

Debug Logging

Debug Logging provides detailed logs from the Intelligence Hub and libraries that support Log4j. Use Debug Logging in cases where detailed logs are required to troubleshoot an issue. For example, this may be an OPC UA connection issue where detailed logs could pinpoint the source of the problem.

Debug Logging should only be used intermittently when troubleshooting an issue. Using it continuously or in production is not recommended as it will impact performance given the large volumes of debug information.

To enable debug logging, enable the Debug Logging option on the Application Settings page. The options are listed from least log messages to most. For example, `Error` shows only error logs, `Info` includes `Error` and `Warning` logs, and `All` includes everything.

Enabling Debug Logging automatically updates the `logback.xml` file in the runtime directory and beings logging to the `runtime.debug.log` file in the runtime directory. To view the logs click the `Debug Log: View` link on the Settings page. From this view you can refresh to see the current logs, download the log file, and reset the logs. Reset deletes the current log file.

An example logback.xml file for logging `Info` logs is provided below. Note the max size of the `runtime.debug.log` is 64MB, at which point the log starts writing over the oldest records. Changing the log level on the Settings page overwrites the logback.xml file and applies the new settings.

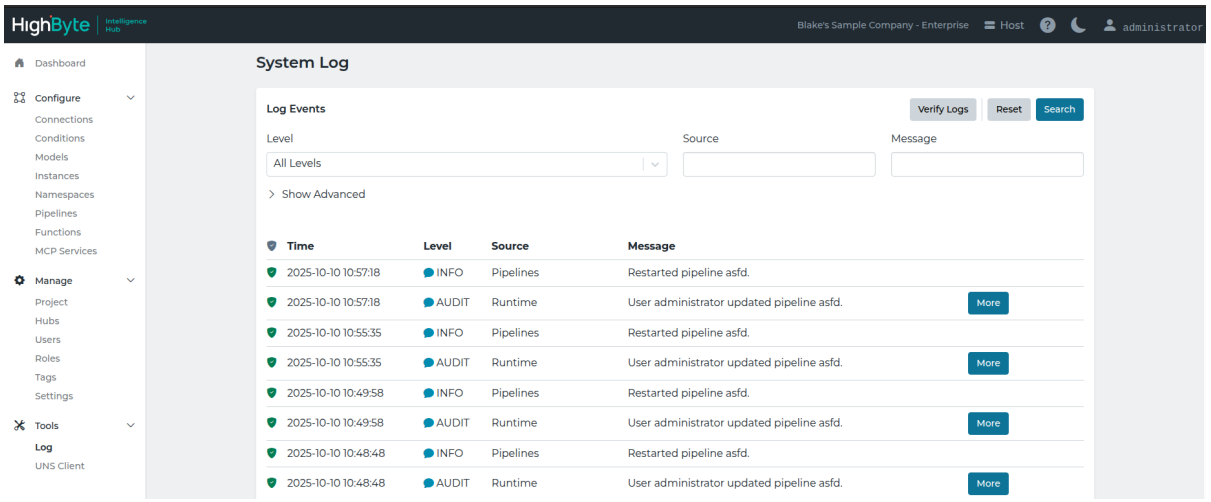
```
<configuration scan="true" scanPeriod="10 seconds">
    <statusListener class="ch.qos.logback.core.status.NopStatusListener"/>
    <appender name="ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>C:\source\intelligencehub_OLD\runtime/runtime.debug.log</file>
        <append>false</append>
        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <fileNamePattern>C:\source\intelligencehub_OLD\runtime/runtime.debug.%i.zip</fil
            <minIndex>0</minIndex>
            <maxIndex>0</maxIndex>
        </rollingPolicy>
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <maxFileSize>64MB</maxFileSize>
        </triggeringPolicy>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-4relative [%thread] %-5level %logger{35}
        </encoder>
    </appender>
    <root level="INFO">
        <appender-ref ref="ROLLING"/>
    </root>
</configuration>
```

Event Log | User Guide

Source: https://guide.highbyte.com/troubleshooting/eventlog/

Event Log

The Event Log is available via the main menu under *Log* and contains the runtime and audit log messages generated by the selected hub. The log is primarily used for troubleshooting connection and pipeline errors.

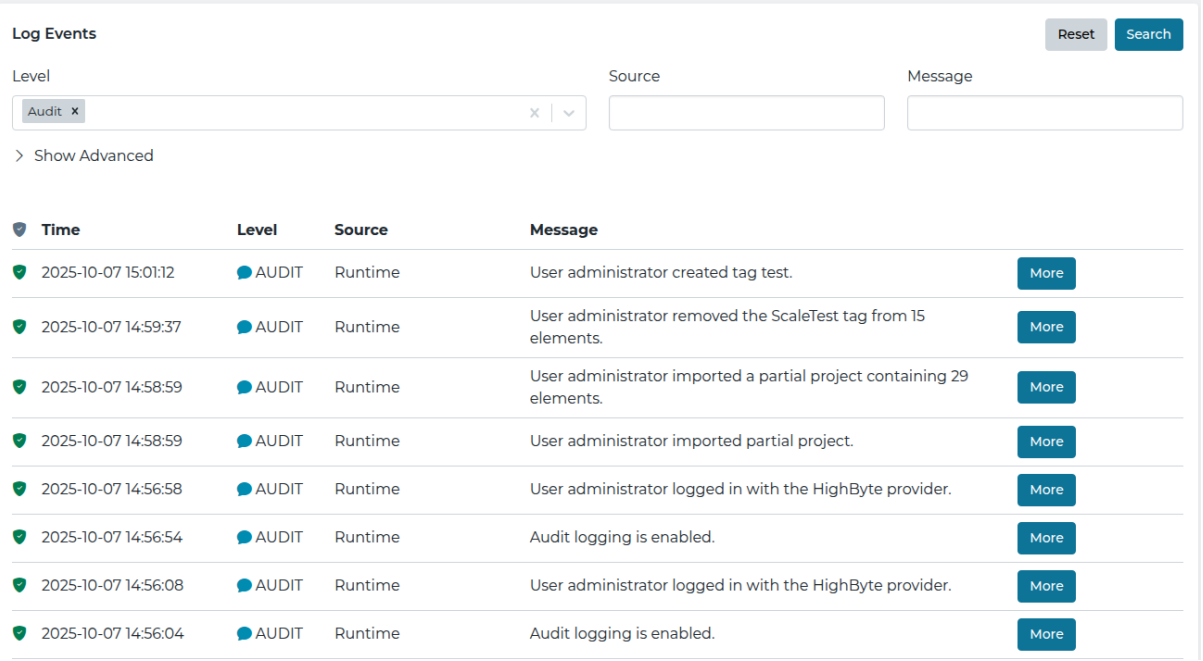


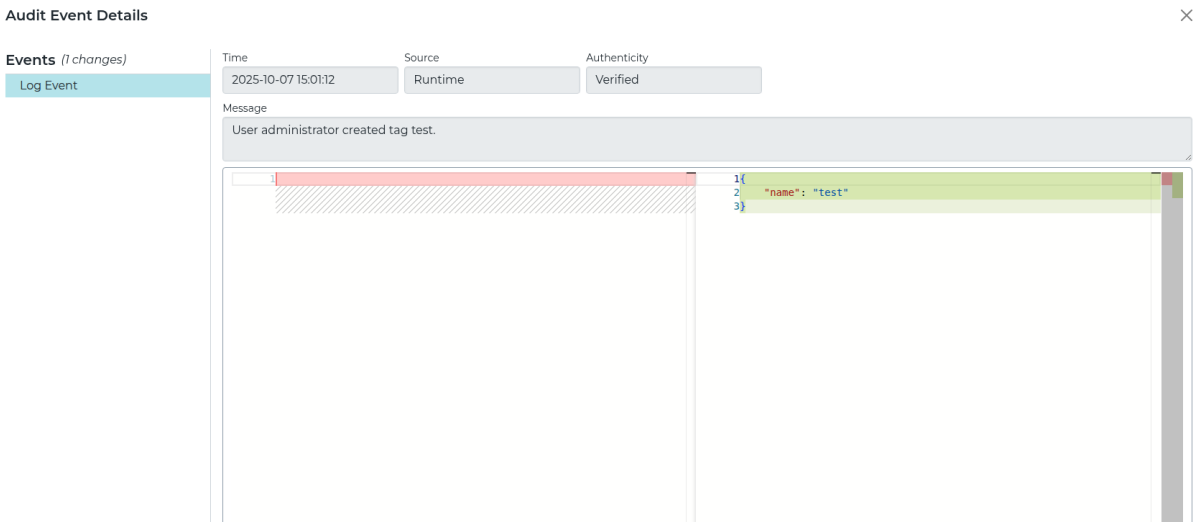
Each log message includes the time (localized), the event level (e.g., info, warning, error, fatal, or audit), the source (the Runtime or a Connector), and the message that can be used for troubleshooting.

The log can easily be searched by the Level, Source, or a general filter that looks at the message body.

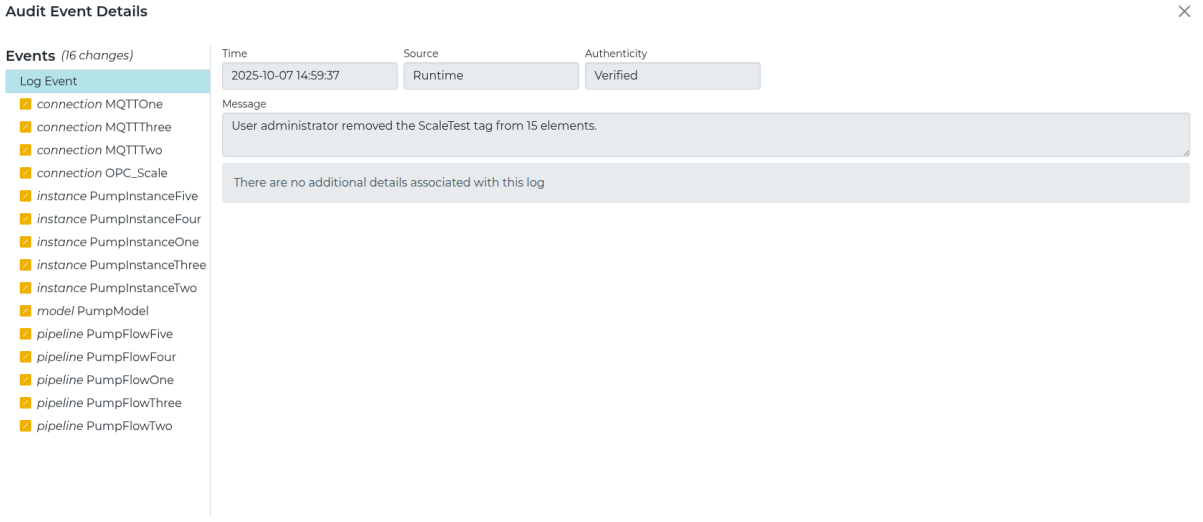
Audit logs additionally have a “more” window which can be used to access a before and after state for the event.

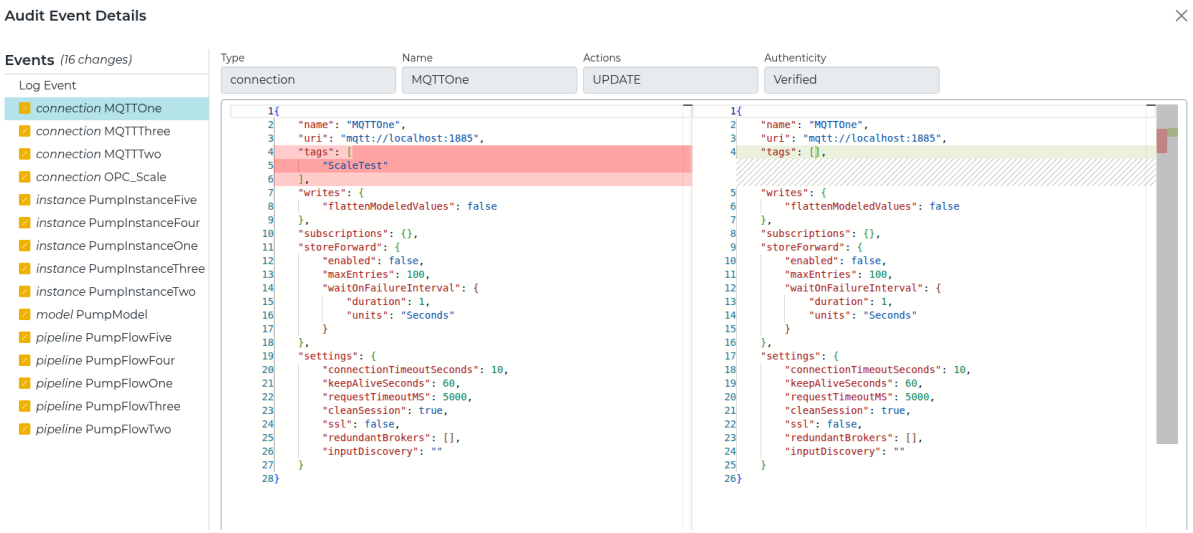
Event, Audit, and detailed log messages are signed upon storage. Messages that are generated by the application instance and unmodified will be marked as verified (green shield) in the log and on event details pages. Use the Verify Logs button to run verification against the full contents of the log. Once executed, users can review the verification summary by choosing View Results.





Additionally, changing some elements may result in additional “cascading” changes to elements which use or reference them (e.g., renaming, tagging, deleting). Logs for these events can be found under the expanded menu for the originating event.





A text-based version of the log is also available in the `intelligencehub-events.log` file (located in the application data directory). Each log message in the text file is in JSON format, allowing for external log monitoring tools to easily consume and index the messages.

Custom Event Messages

It's common to use applications like Splunk or DataDog to monitor the text based log to detect errors. In these cases it can be useful to create custom log messages, based on specific logic you are monitoring. `{{System.EventLog.Info}}` can be used as a pipeline write messages to the log. Format the writes as a string value.

As an example, a pipeline can monitor the number of errors on a given set of pipelines and write to the logs if the number of errors exceeds 10.

Debugging Pipelines | User Guide

> Source: https://guide.highbyte.com/troubleshooting/pipelines/

Debugging Pipelines

Tools for Debugging Pipelines

Pipelines have multiple tools that can be used to assist with troubleshooting/debugging.

All of these tools are available from the Pipeline Graph and include the following:

Tools	Details
Status	Monitor the status of stages in realtime
State	View the current state of pipeline metadata variables
Statistics	View pipeline and stage level statistics to monitor performance
Debug	Debug a pipeline in a safe environment and inspect events, and state changes in detail
Replay	Replay previous executions of the pipeline and inspect their events in detail

Status & Metrics | User Guide

> Source: https://guide.highbyte.com/troubleshooting/status_and_metrics/

Status & Metrics

The Intelligence Hub provides visual feedback of the status of Connections and Pipelines and also makes this status information and performance metrics available as source data in the system. This allows for triggering using status and metric changes, as well as making this data available to monitoring applications.

This section covers the status and metric information available for each object.

Connection Status

Connection status is reported through the main *Connections* view, as well as on the *Details* tab for each connection.

A Connection's status is set to *Good* when the connection is successfully established. *Note: Some connection types, like file connections are always Good as there is no physical connection.*

A Connection's status is set to *Bad* when the connection is not successfully established (e.g., MQTT server cannot be reached, HTTP server is not available or authentication fails).

The Connection status also shows the last error message for the Connection. When a connection transitions from a *Bad* state to *Good* state a message is logged to the event log. Select the *Log* menu option to view more details on the error.

Connections expose status and metric information under the `System.Connections.<connectionName>` reference. Below is the information exposed for each connection.

Metric	Description
name	The name of the connection.
lastError	The last error message produced by the connection.
healthy	True if the connection is in a good state, false if it's in an error state.
storeAndForwardEnabled	True/False if store and forward is enabled for the Connection.
status	The status string of the connection (Good or Bad).
statistics.pendingWrites	The number of writes queued to send. When Store & Forward is enabled, this shows how many writes are pending to send.
statistics.droppedWrites	In the event that the write queue fills up for in memory or store and forward, and new writes are dropped, this metric is a count of how many writes are dropped since the last time the connection was established.
statistics.outputs	An array of all the outputs and the pending and dropped writes for each output.
statistics.totalSizeBytes	The total allocated space for pending writes.

Pipeline Status

Pipeline status is reported through the Status and Replay tools in the Pipeline Graph.

Like Connection Status, Pipeline Status includes Good/Bad status, last error messages etc. In addition to that, Pipelines also provide the same status information for each Stage.

A Pipeline is in an error state if any of its Stages were in an error state on the last Execution of the Pipeline, and in Good status if there were no errors on the last execution.

A Pipeline Stage is in an error state if any event produced an error in the last Execution of the pipeline. Previous executions can be inspected in detail using the Replay tool.

Pipeline/Stage status and metric information is available under the `System.Pipelines.<pipelineName>` reference. Below is the information exposed for each pipeline.

Metric	Description
name	The name of the pipeline.
lastError	The last error message produced by the pipeline.
state	Good or Bad.
healthy	True if the connection is in a good state, false if it's in an error state.
status	The status string of the connection (Good or Bad).
stageStatus	An array of the status objects for each stage. See below table for details.
statistics	An object with statistics for the pipeline executions. See below table for details

Stage Status Metric	Description
name	The name of the stage.
lastError	The last error message produced by the stage.
pipelineName	The name of the parent pipeline.
state	Good or Bad.
healthy	True if the connection is in a good state, false if it's in an error state.

Pipeline Statistics	Description
idle	True if the pipeline is not processing any events. False otherwise.
totalRuns	The total number of events processed by the pipeline
runsPerSecond	The number of runs per second being completed by the pipeline. This is an instantaneous rate based on the runs completed in the last 5 seconds.
totalErrors	The total number of pipeline execution errors.
errorPercentage	The ratio of totalErrors to totalRuns expressed as a percentage.
queuedWrites	The number of writes waiting to be processed by the pipeline.
lastCompleteTime	The last time the pipeline completed processing an event.
executingSamples	An object containing the mean, min, max, standard deviation, and sample count. (since the last time the pipeline was saved)
waitingSamples	An object containing the mean, min, max, standard deviation, and sample count. (since the last time the pipeline was saved)
stages	An array that contains statistics for each of the pipeline's stages. See below table for details.

Stage Statistics	Description
stageName	The name of the stage.
totalRuns	The total number of events processed by the stage.
totalErrors	The total number of stage event errors.
lastCompleteTime	The last time the stage completed processing an event.
executingSamples	An object containing the mean, min, max, standard deviation, sample count. (since the last time the pipeline was saved)

Internal Sources

The hub also exposes internal sources that are not linked to configuration objects. These are available under the `System.Internal.<name>` reference and described below.

Internal Source	Description
DateTime	The current UTC time of the server in ISO8601 UTC format.