CONNEX

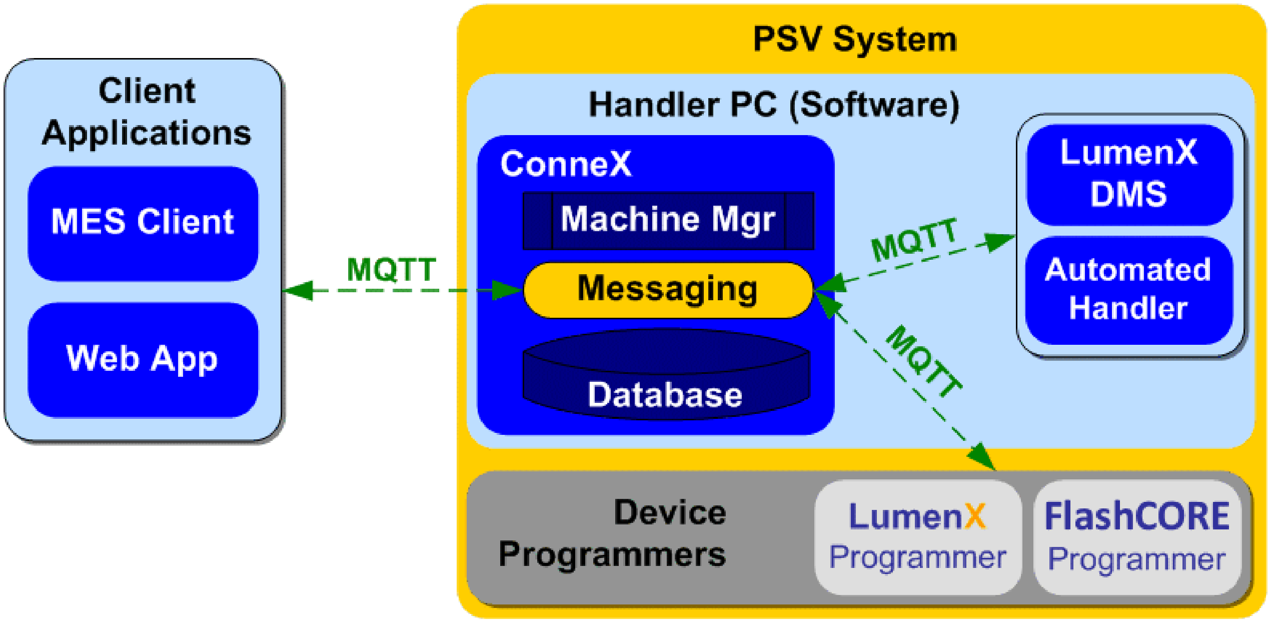
Table of Contents

Introduction

ConneX 3.0 Introduction

In addition to the job auditing and manufacturing traceability capabilities (delivering device programming results through customizable XML templates) provided in prior releases, the latest ConneX version introduces an Event Model that you can access through the new ConneX API (application programming interface) to expand the accessibility and scale of your programming operations data.

The latest ConneX leverages the Message Queuing Telemetry Transport (MQTT) protocol to exchange event messages between both internal PSV system components and external client applications.



Internally, ConneX handles all of the data collection and event retrieval from underlying system components (ex. Automated Handler, Programmers, and LumenX Data Management Software) and exposes them through an external API with a Publisher-Subscriber model: consumers (ex. a custom dashboard/Web or MES application) call a specific ConneX API to register a callback URL that essentially subscribes them to the desired event--each time the event occurs, ConneX notifies each consumer who registered/subscribed for event notification.

The ConneX API operates asynchronously to essentially detach internal system functions and events from external applications and consumers, which in turn simplifies the development skills required to integrate your programming operations data with your MES or custom Web application because developers need not know or learn the intricacies of how PSV Systems operate. API Consumers simply need to create an HTTP URL (with some query string parameters to fetch and filter desired event fields).

Template Records/Manager is the component in ConneX that transforms ConneX data and delivers it through standard templates (such as .XML and .JSON), making it easier for customers to view and export raw records using a template for storage on disk.

From custom real-time dashboards for programming operations to data integration with MES applications, the latest ConneX provides increased visibility and control over your programming operations to reduce the burden and cost of managing your programming systems. The latest ConneX maintains backwards-compatibility for existing ConneX users who are now collecting and analyzing their programming data but also makes it easier for all users to access more data because it features a commonplace yet flexible Web service over standard HTTPS as the API Provider.

The ConneX API conforms with the MQTT Version 5.0 OASIS Standard, which you can reference at https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

Note
The ConneX API returns MQTT data as a real-time feed of Automated Handler, LumenX/FlashCORE Software, and Programmer events (which are all persisted by Machine Manager in the Audits database for ConneX). Therefore, if any disruption occurs (ex. ConneX service goes offline during a job) then events may be lost and retry functions must be implemented for MQTT (and/or any missed events must be retrieved from the GraphQL API, which provides historical data).

Requirements

This section describes the software requirements for running Machine Manager and ConneX Server.

Machine Manager Requirements

To use the latest ConneX version, each Machine Manager computer must meet the following minimum requirements.

Software Application	Version	Notes
Windows 10 operating system	Pro	Either 32-bit or 64-bit
ConneX Machine Manager	Latest	Supports x86 and x64
Automated Handler (AH700/CH700)	3.3.x	keep IP address as 127.0.0.1; default Port is TCP 9002

For more information about Automated Handler software, see Automated Handling software.

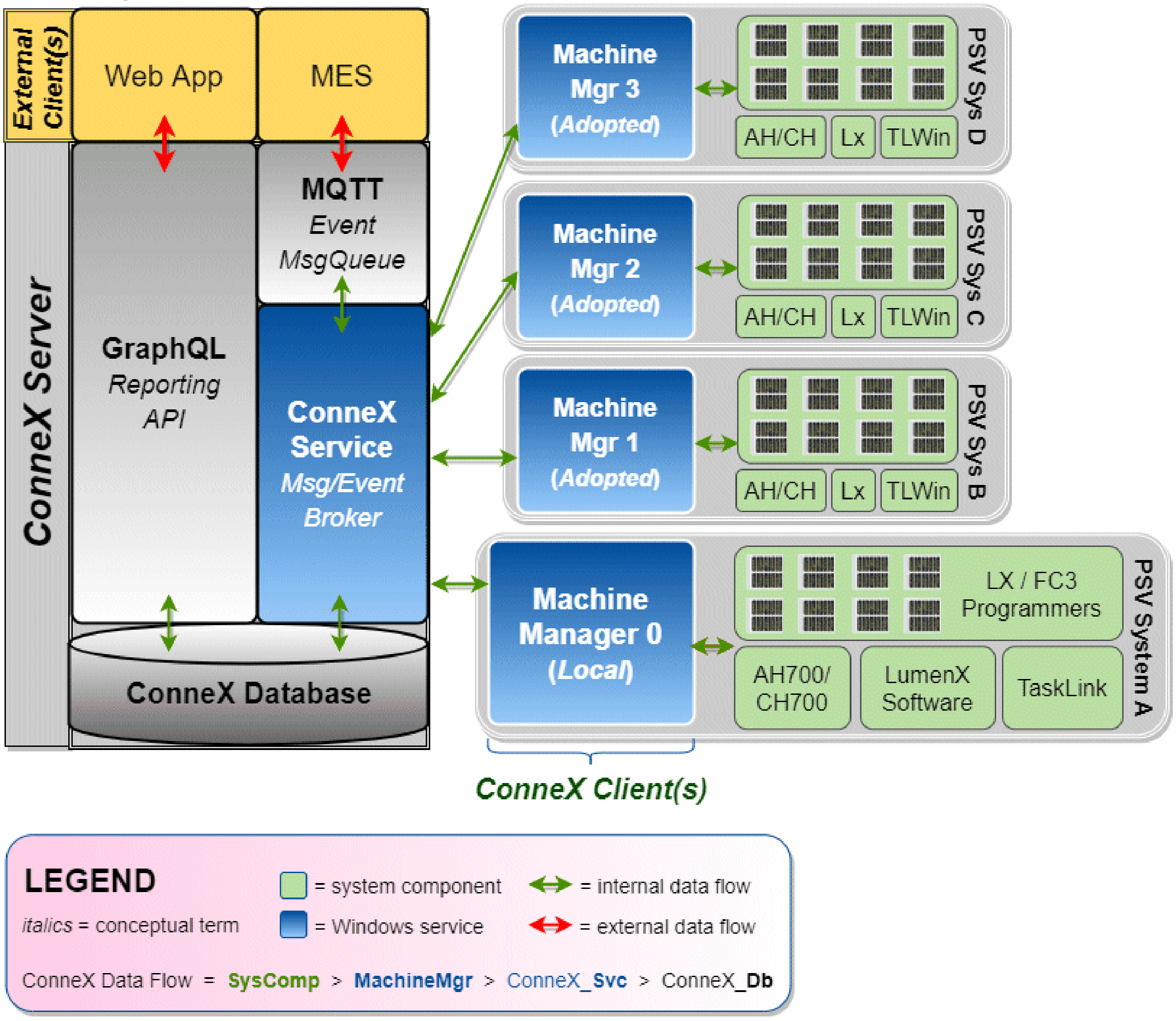
ConneX Server Requirements

To use the latest ConneX version, the ConneX Server must meet the following minimum requirements.

Software Application	Version	Notes
Windows 10 operating system	Pro	x64 only
or Windows Server 2016/2019/2022		x64 only
ConneX Server	Latest	x64 only
Data I/O License Manager	1.0.0.61	requires .NET Framework 4.7.2
Chrome browser	100.0.x	browser cookies required for user login/authentication (supports either **Allow all cookies** or **Block third-party cookies**; does not support **Block all cookies**); Internet Explorer NOT supported

Installation

This page provides step-by-step instructions on installing the latest ConneX version and establishing proper communication between each Machine Manager instance and the ConneX Server.



The latest ConneX version provides separate installers for **ConneX Server** and **Machine Manager**: install ConneX Server on a centralized 'data reporting' computer, then install Machine Manager on each 'client' computer (essentially, the Handler/Host PC in a PSV System). This separation is designed to optimize the management and handling of data and event collection from system components. By running ConneX Server on a separate PC, data can be aggregated and analyzed in a central location while the Machine Manager(s) focuses on data aggregation and event messaging between ConneX and various PSV system components.

Before proceeding with installation, ensure the computer(s) meet the ConneX requirements as described at the bottom of the Introduction page.

Machine Manager Installation

Run the setup wizard for Machine Manager to install the Machine Manager Service:

- **ConneX Machine Manager Service**, for handling data aggregation and event messaging between multiple PSV Systems.

Machine Manager requires specific network ports to be open for proper communication with ConneX Server; the setup wizard for Machine Manager automatically opens the following ports in Windows Defender Firewall:

Service/Inbound Rule	**Required Port**
Machine Manager gRPC port	TCP 5000
Machine Manager LumenX Discovery Port Receive	UDP 9081
Machine Manager LumenX WCF port	TCP 9000
Machine Manager MQTT port	TCP 9002
Machine Manager SSDP port	UDP 1900

Note

If you change the MQTT port(s), then also restart the ConneX Machine Manager Service (see below).

After installing Machine Manager, reboot and double-check that the Machine Manager service is running on the PSV Handler/Host computer:

1. Click **Start** > type ***Run*** (hit ENTER) > *services.msc*
2. In the Services window, start the following service if needed (right-click > **Start**):
    - ConneX Machine Manager Service
3. If the Machine Manager Service is not already started (and set to start automatically):
    - Right-click the service
    - Select **Start**
    - Right-click the service again
    - Click **Properties**
    - On the **General** tab, from the **Startup Type** drop-down list, select **Automatic**

Note
Beyond this point, you remotely manage each Machine Manager instance/client/computer from the ConneX Server because Machine Manager is essentially "headless" (requires no user interface).

Handler Software Configuration

To configure the "AH700/CH700" handler software of a PSV System (Machine Manager instance) for ConneX, keep the same IP address of **127.0.0.1** with default port **TCP 9002**. If you change the Machine Manager port, then also reflect the same change in the 'WinAH400.ini' file.

1. On the Handler/Host PC of a PSV System (Machine Manager instance), open File/Windows Explorer to the following directory:
    - If Machine Manager is running on a PSV7000, open 'C:\AH700'
    - If Machine Manager is running on a PSV3000/3500/5000, open 'C:\CH700'
2. Locate the `WinAH400.ini` file and open it with a text editor (such as Notepad).
3. Find the following lines in the file. If they are not present, then add them:

    ini

    ```
    CheckConneX=true
    ConneX3=true IP=127.0.0.1 Port=9002
    ```

4. Save the change(s).

ConneX is now set to communicate with the respective handler software (AH700 or CH700) over the specified IP address and port number.

ConneX Server Installation

Run the setup wizard for ConneX Server to install the main ConneX Service and Data I/O License Manager:

- **ConneX Service**, the primary ConneX handler service for managing data and event collection from system components.
- **Data I/O License Manager**, for validating the propriety of authorized Data I/O software products, features, and services.

The centralized ConneX Server requires specific network ports to be open for proper communication with each Machine Manager instance; the setup wizard for ConneX Server automatically opens the following ports in Windows Defender Firewall:

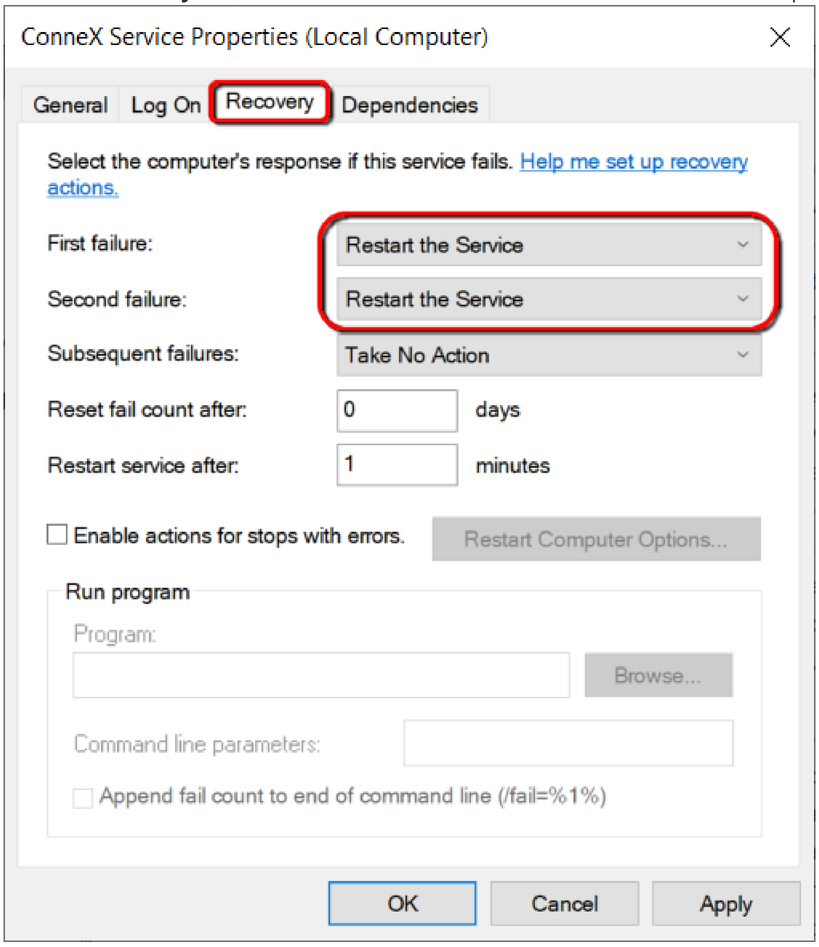
Service/Inbound Rule	**Required Port**
ConneX internal MQTT port	TCP 9002
ConneX MQTT port	TCP 1883
ConneX SSDP port	UDP 1900
ConneX web port	TCP 5001
License Manager Discover Port Receive	UDP 9081
License Manager gRPC Service Port	TCP 9003
License Manager WCF Service Port	TCP 9000
License Manager Web Port	TCP 5002
LicenseManagerServer	UDP any
LicenseManagerServer	TCP any
Template Manager MQTT port	TCP 1883
Template Manager web port	TCP 5004

Note
If you change the MQTT port(s), then also restart the ConneX Machine Manager Service (see below).

After installing ConneX Server, reboot and double-check that the ConneX services are running:

1. Click **Start** > type ***Run*** (hit ENTER) > *services.msc*
2. In the Services window, start the following services if needed (right-click > **Start**):

   - ConneX EventStoreDB
   - ConneX PostgreSQL + TimescaleDB Service
   - ConneX Service
   - DataIO License Manager Service
3. If any of the four ConneX services are not already started (and set to start automatically):
   - Right-click the service
   - Select **Start**
   - Right-click the service again
   - Click **Properties**
   - On the **General** tab, from the **Startup Type** drop-down list, select **Automatic**
4. Skip this step if installing ConneX Server on a Windows 10 computer. Else for Windows Server 2016/2019/2022:
   - Click the **Recovery** tab, then set both **First failure** and **Second failure** to the option **Restart the Service**.



ConneX Service Properties (Local Computer)

General | Log On | Recovery | Dependencies

Select the computer's response if this service fails. Help me set up recovery actions.

First failure: Restart the Service

Second failure: Restart the Service

Subsequent failures: Take No Action

Reset fail count after: 0 days

Restart service after: 1 minutes

☐ Enable actions for stops with errors. Restart Computer Options...

Run program

Program:

Browse...

Command line parameters:

☐ Append fail count to end of command line (/fail=%1%)

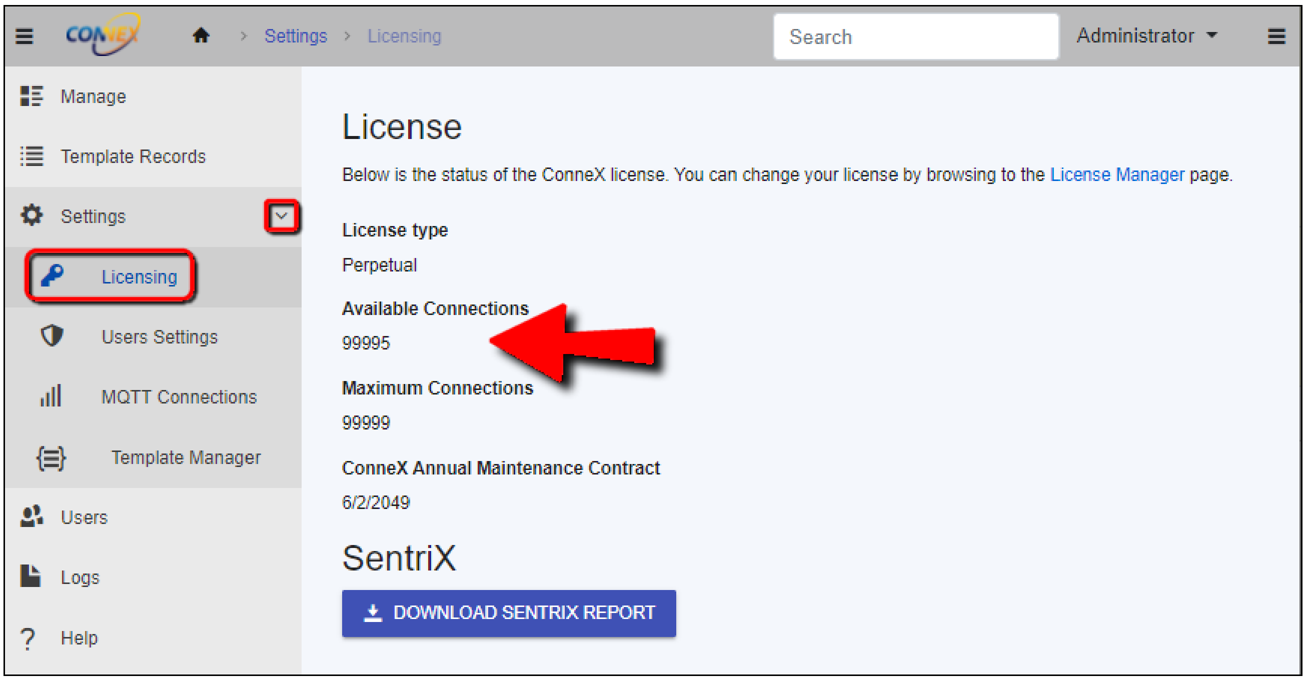
OK Cancel Apply

   - Click **Apply** and then **OK**.

Activate ConneX License

After installing Machine Manager and ConneX Server, ensure ConneX has sufficient licenses available to satisfy each Machine Manager instance/connection.

1. In the left pane, click **Settings**, then click **Licensing**.
2. On the **License** page, ensure **Available Connections** is greater than one (and/or equal to the number of expected Machine Manager connections).



License

Below is the status of the ConneX license. You can change your license by browsing to the License Manager page.

License type
Perpetual

Available Connections
99995

Maximum Connections
99999

ConneX Annual Maintenance Contract
6/2/2049

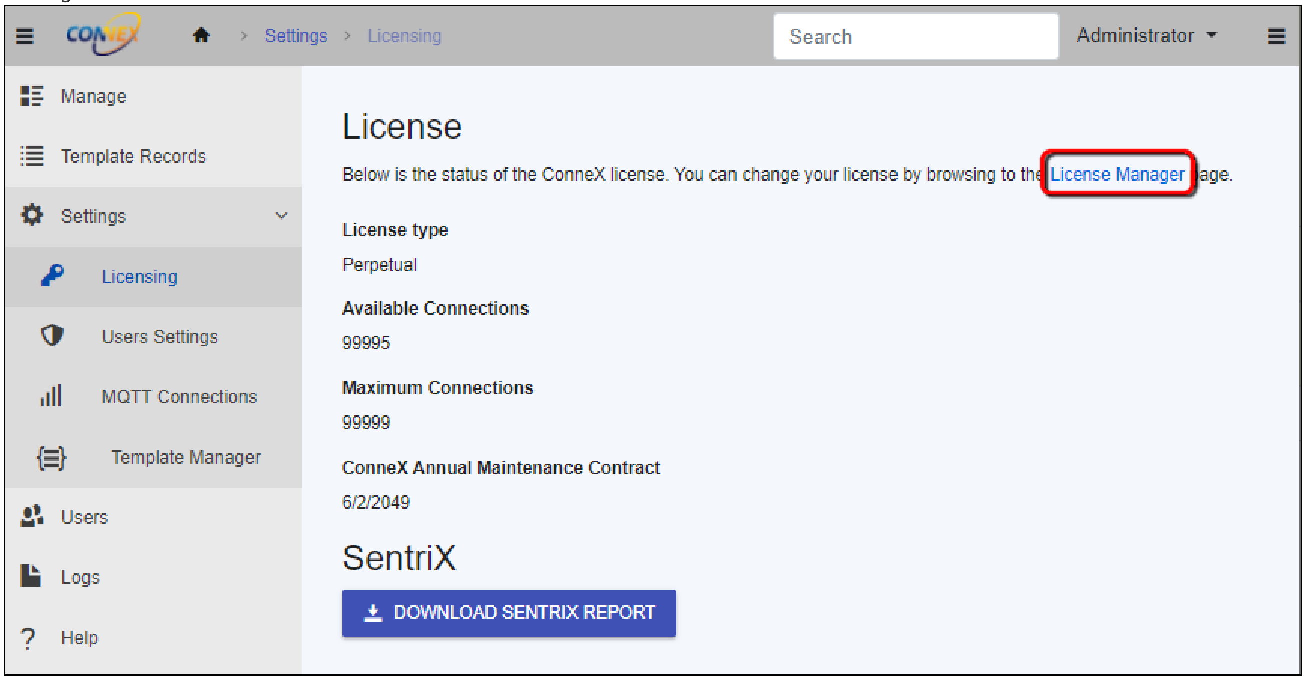
SentriX

DOWNLOAD SENTRIX REPORT

Note

You may need to refresh the License page to display any modifications to ConneX licensing.

3. If **Available Connections** shows unexpected information and/or to access Data I/O License Manager directly, click the License Manager link.



License

Below is the status of the ConneX license. You can change your license by browsing to the License Manager age.

License type
Perpetual

Available Connections
99995

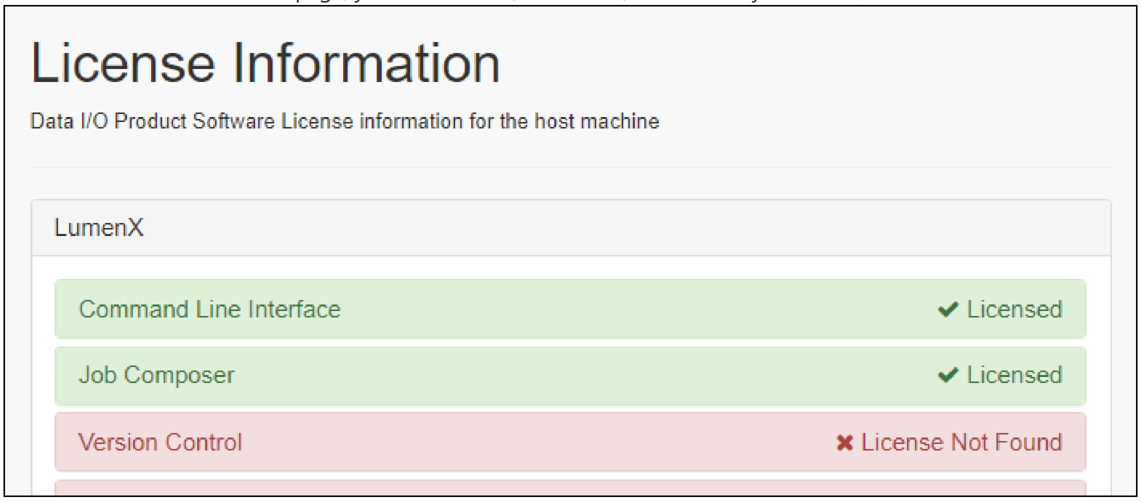
Maximum Connections
99999

ConneX Annual Maintenance Contract
6/2/2049

SentriX

DOWNLOAD SENTRIX REPORT

4. On the **License Information** page, you can activate, deactivate, and refresh your Data I/O software licenses.



License Information

Data I/O Product Software License information for the host machine

LumenX

Command Line Interface	✓ Licensed
Job Composer	✓ Licensed
Version Control	✖ License Not Found

Note

You may need to refresh the License Information page to display any modifications to ConneX licensing.

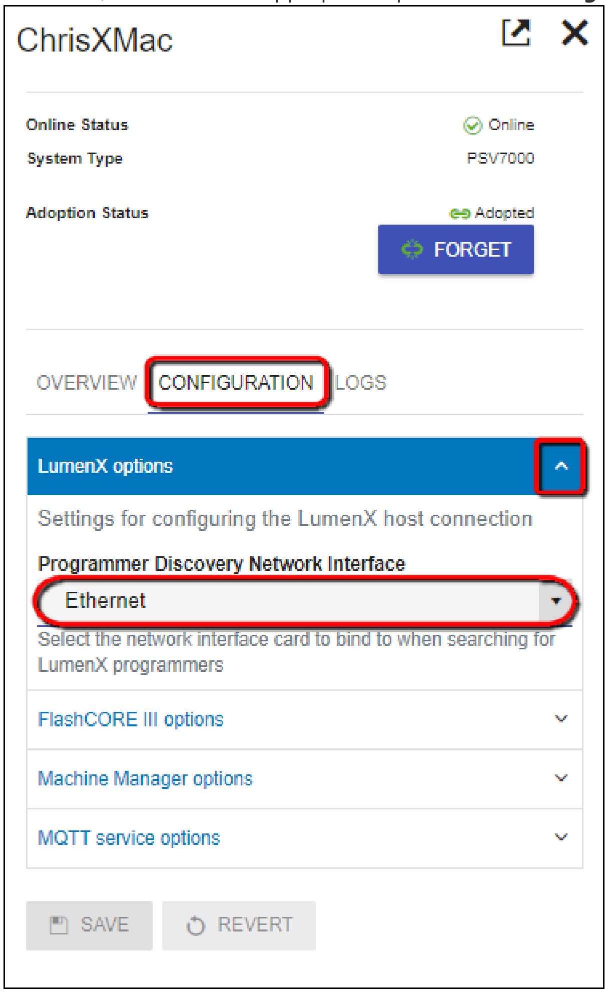
With your Machine Manager(s) and ConneX Server installed, proceed to setup the programmers as described in the Configuration steps.

Configuration

This page provides instructions for configuring the programmers of a PSV System (Machine Manager instance) from the ConneX Server. To establish connectivity between ConneX Server and each Machine Manager/PSV System, complete the steps under Adding a New System.

Configure Programmer Interface

1. On the **Manage** page, select a PSV System to configure.
2. In the system tile, click the **Configuration** tab.
3. Now select the appropriate network interface for programmer communication:
   - For LumenX, click the **LumenX options** drop-down arrow, then select the appropriate option from the **Programmer Discovery Network Interface** drop-down list.



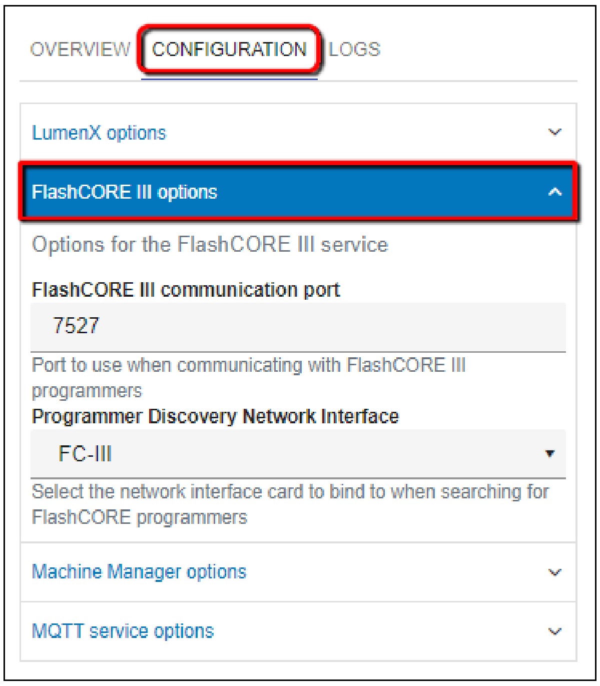
Settings for configuring the LumenX host connection

Programmer Discovery Network Interface: Ethernet

Select the network interface card to bind to when searching for LumenX programmers

   - For FlashCORE, click the **FlashCORE III options** drop-down arrow, then select the appropriate option from the **Programmer**

Discovery Network Interface drop-down list.



Options for the FlashCORE III service

FlashCORE III communication port

7527

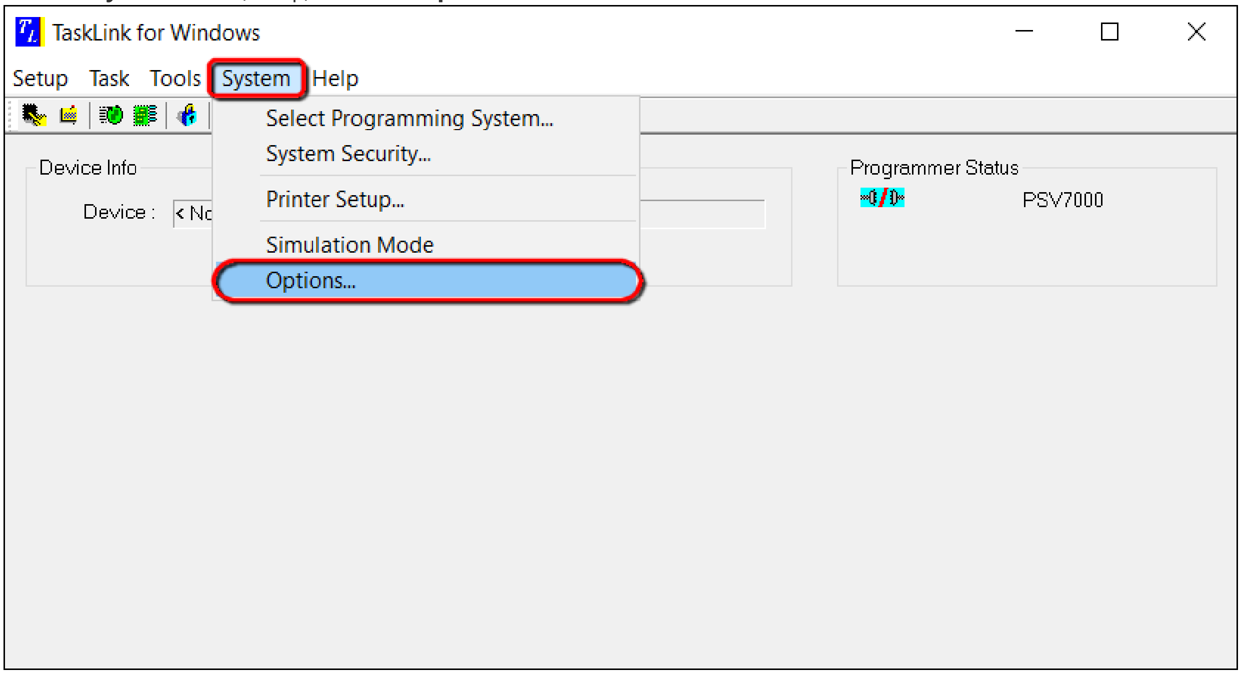
Port to use when communicating with FlashCORE III programmers

Programmer Discovery Network Interface

FC-III

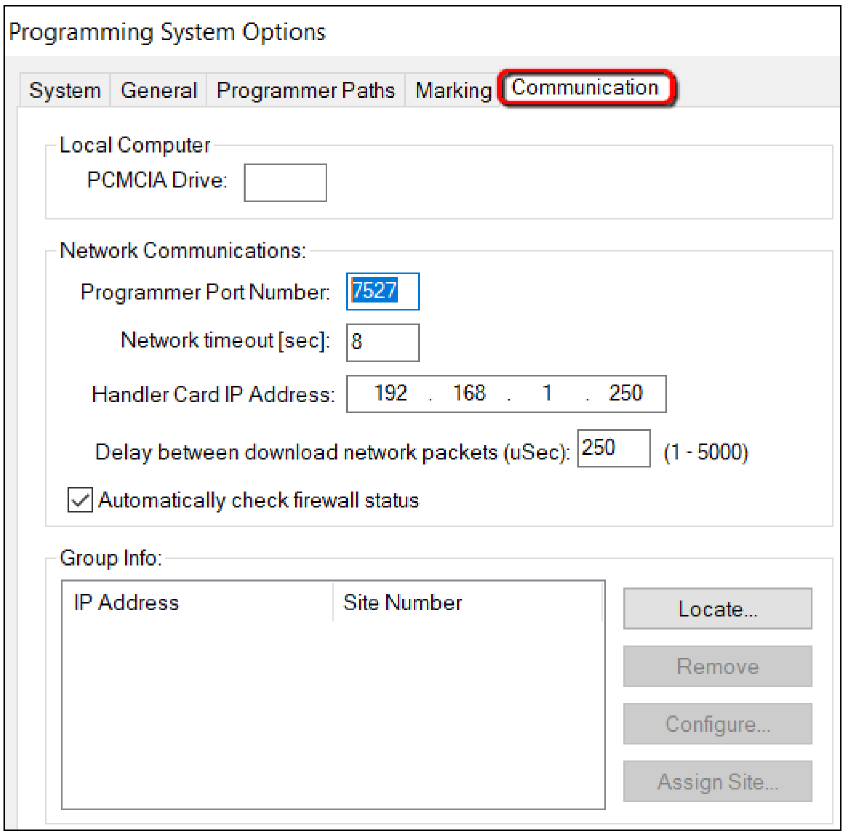
Select the network interface card to bind to when searching for FlashCORE programmers

4. For FlashCORE only (skip this step for LumenX): Ensure the **FlashCORE III communication port** value (default is 7027) matches the programmer port number configured in TaskLink:
   - Start TaskLink for Windows
   - Click the **System** menu (at top), then click **Options**.



   - In the **Programming System Options** dialog box, click the **Communication** tab, and then confirm the **Programmer Port**

Number matches the value in ConneX.


Programming System Options

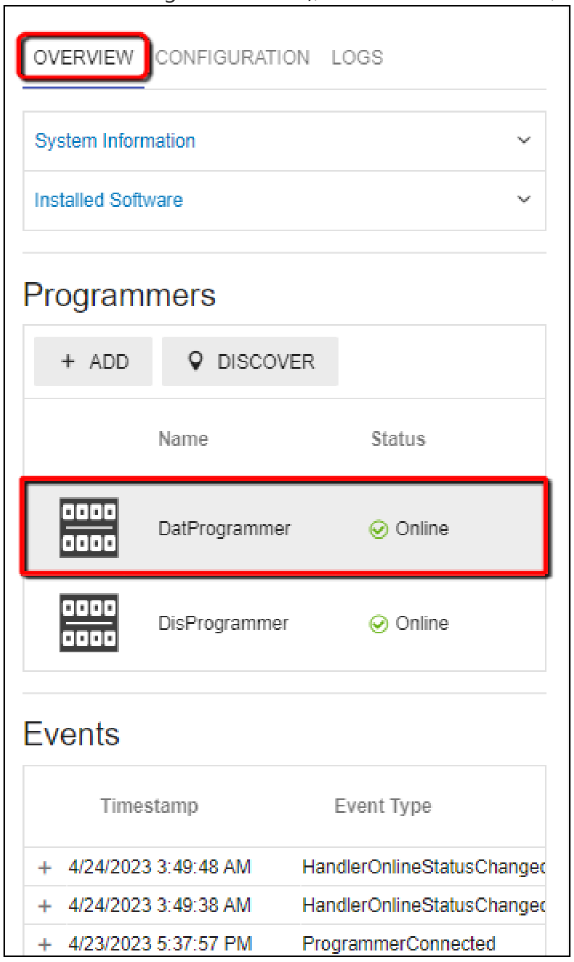
Local Computer
PCMCIA Drive:

Network Communications:
Programmer Port Number: 7527
Network timeout [sec]: 8
Handler Card IP Address: 192 . 168 . 1 . 250
Delay between download network packets (uSec): 250 (1 - 5000)
[✓] Automatically check firewall status

Group Info:

IP Address	Site Number

5. To change the name of a programmer (which ConneX writes to logs and records), click the **Overview** tab, then select the desired programmer (under the **Programmers** section).



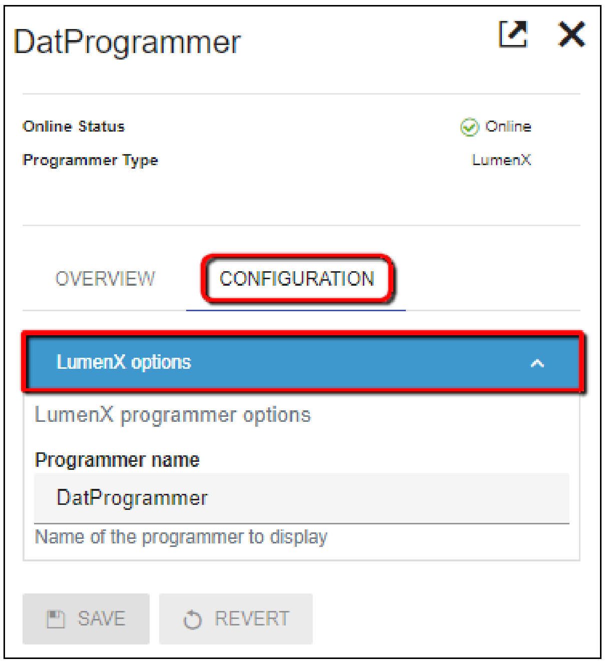
Programmers

Name	Status
DatProgrammer	Online
DisProgrammer	Online

Events

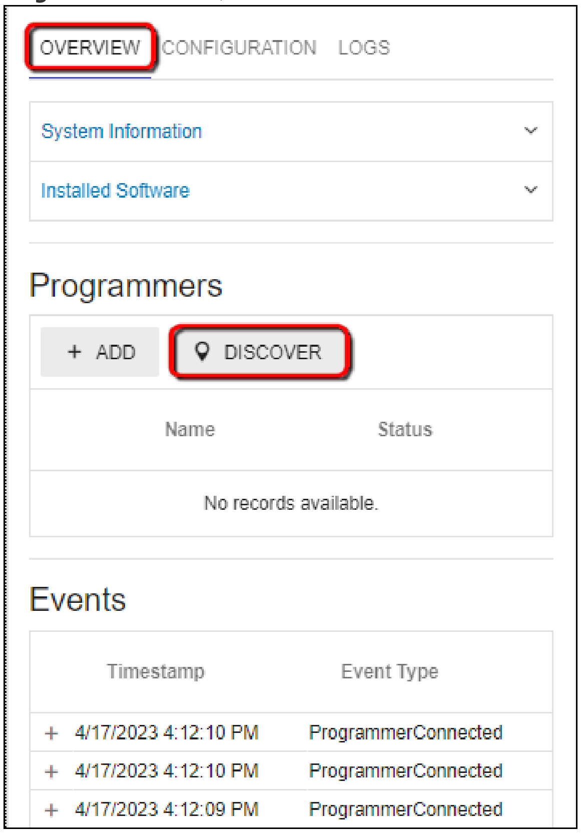
Timestamp	Event Type
4/24/2023 3:49:48 AM	HandlerOnlineStatusChanged
4/24/2023 3:49:38 AM	HandlerOnlineStatusChanged
4/23/2023 5:37:57 PM	ProgrammerConnected

6. In the programmer tile, click the **Configuration** tab, then expand the **LumenX options** (or **FlashCORE III options**) drop-down list.



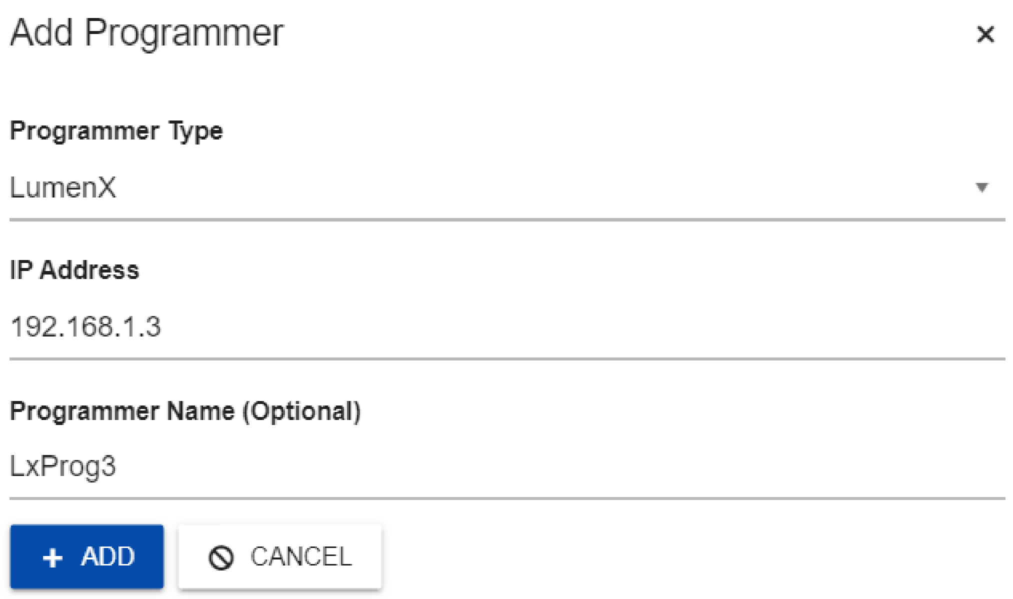
Check Programmer Connectivity

1. Return to the system tile (and its **Overview** tab), then under the **Programmers** section, confirm that the list shows the programmer(s) you intend to use for ConneX; else click **Discover**.



Timestamp	Event Type
+ 4/17/2023 4:12:10 PM	ProgrammerConnected
+ 4/17/2023 4:12:10 PM	ProgrammerConnected
+ 4/17/2023 4:12:09 PM	ProgrammerConnected

2. If the desired programmer does not appear in the list after running automatic discovery, then click **Add** (next to **Discover** button).
3. In the **Add Programmer** window, select the **Programmer Type** (FC or LX), specify its **IP address**, type a **Programmer Name** (optional), and then click **Add**.

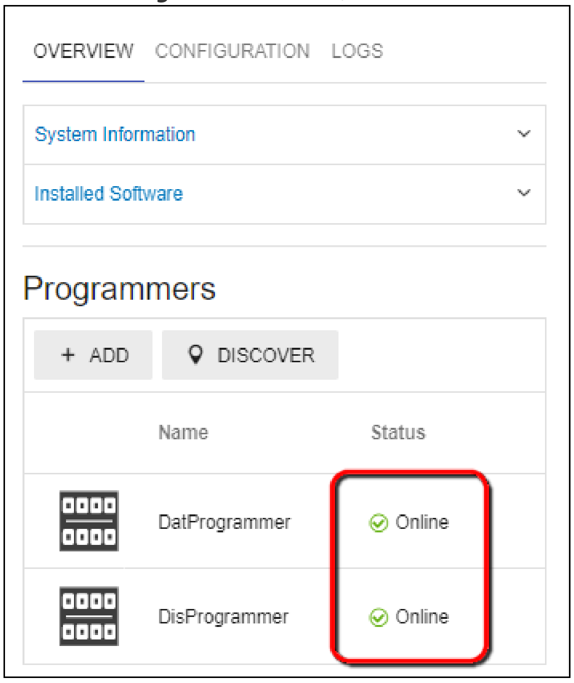

Add Programmer

Programmer Type: LumenX

IP Address: 192.168.1.3

Programmer Name (Optional): LxProg3

4. Now in the **Programmers** section, confirm that the added programmer appears in the list with a **Status** of **Online** (green color).

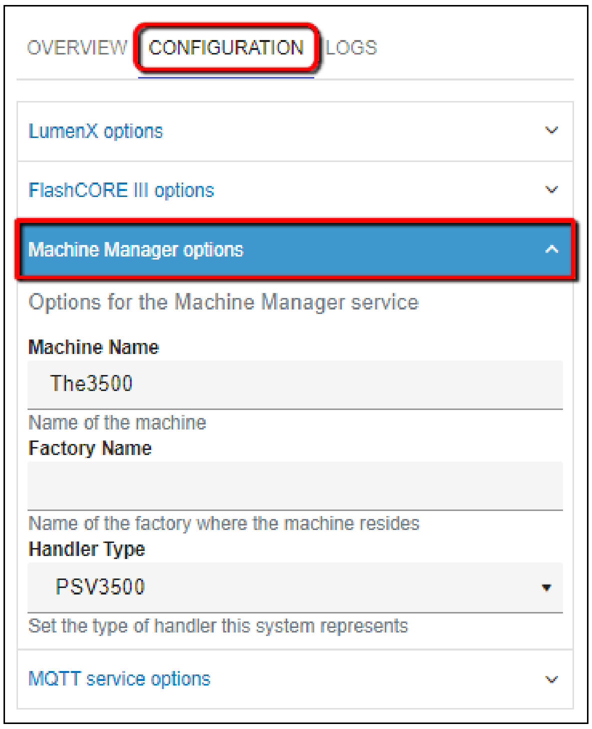


Name	Status
DatProgrammer	Online
DisProgrammer	Online

Configure Machine Manager options

To change the Machine Name, Factory Name, or Handler Type (all of which ConneX writes to logs and records):

1. On the **Manage** page, select a PSV System/Machine Manager to configure.
2. In the system tile, click the **Configuration** tab, then expand the Machine Manager options drop-down list.



Options for the Machine Manager service

Machine Name: The3500
Name of the machine

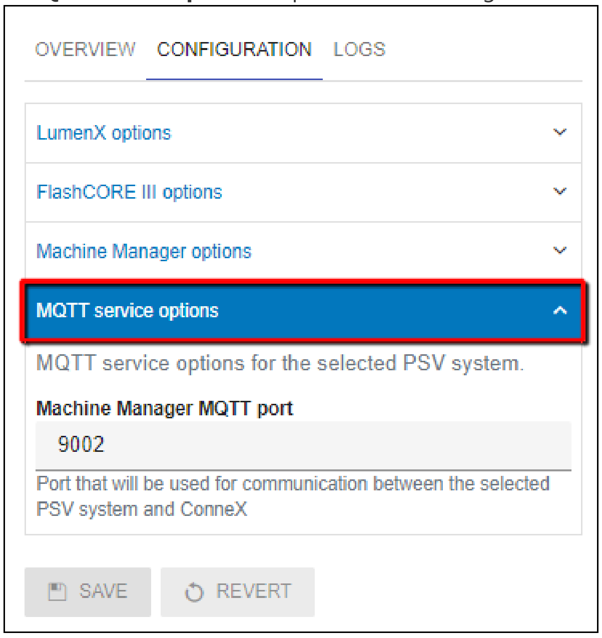
Factory Name:
Name of the factory where the machine resides

Handler Type: PSV3500
Set the type of handler this system represents

3. Click **Save**.

Configure MQTT service options

Similarly, click the **MQTT service options** drop-down list to configure the **Machine Manager MQTT port** (default is TCP port 9002), and then click **Save**.



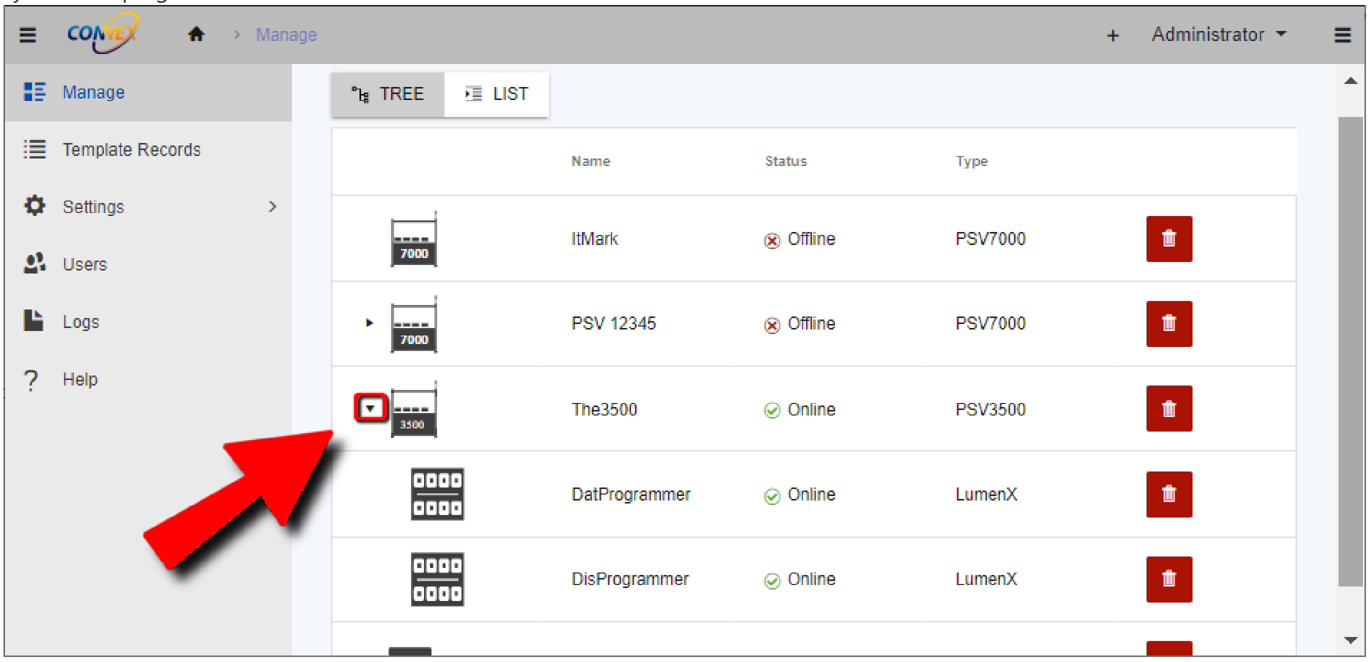
MQTT service options for the selected PSV system.

Machine Manager MQTT port: 9002
Port that will be used for communication between the selected PSV system and ConneX

Manage

The **Manage** page displays all entities (such as systems, handlers, programmers, and associated programs) for the current instance. The system and programs are organized in two views: tree view and flat view.

In the tree view, the systems are listed first and will expand, making it easy to see the hierarchical relationship between systems and their associated programmers. This view is useful for understanding the layout of your programming systems and navigating quickly to the system and programmer of interest.



Name	Status	Type
ItMark	Offline	PSV7000
PSV 12345	Offline	PSV7000
The3500	Online	PSV3500
DatProgrammer	Online	LumenX
DisProgrammer	Online	LumenX

In contrast, the flat view displays all systems and their associated programmers on the same level, making it easier to see all the systems and programmers at once. This view is helpful when searching for a particular system or programmer and is a more condensed representation of the programming systems.

Entity List

The manage grid shows the following for each entity (essentially a PSV System or programmer):

Column	Description
Name	The given name of the entity.
Online Status	Indicates the status of the entity (Online or Offline).
Type	Indicates the type of the entity (ex. PSV7000, PSV5000, etc.).
Adoption Status	Indicates the adoption status of the system.

Online Statuses

Entities (PSV Systems or programmers) can have the following online statuses:

Status	Description
Online	The entity is online and is operational (sending information to ConneX Server)
Offline	The entity is offline and is not operational (NOT sending data to ConneX Server)

Types

Systems can have the following types:

Type	Description

Type	Description
Programming System types	• Desktop • PSV3000 • PSV3500 • PSV5000 • PSV7000
Programmer types	• FlashCORE III • LumenX

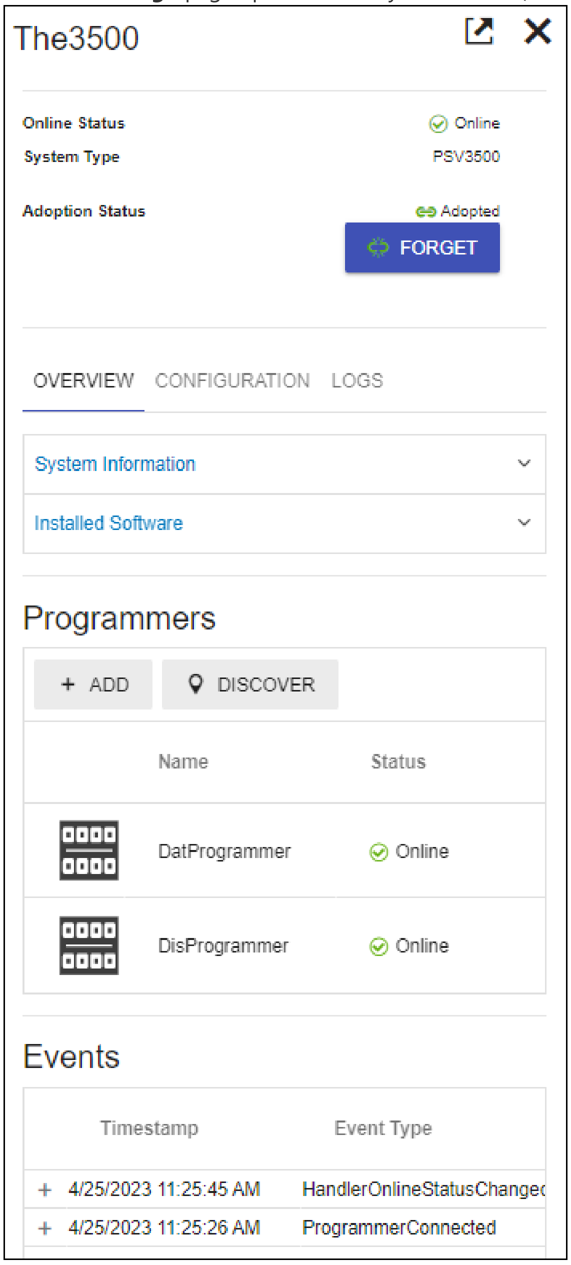
Adoption Statuses

Systems can have the following adoption statuses:

Type	Description
System adoption status	• **Adopted**: System is currently adopted and in use. • **Not Adopted**: System is NOT adopted and is available for adoption. • **Adopted by other**: System is adopted and in use by another entity.

Configuring an Entity

Clicking a row on the **Manage** page opens the Entity sidebar/tile (in the right pane), which allows you to view configuration and entity information.



The3500

Online Status: Online
System Type: PSV3500
Adoption Status: Adopted

Programmers

Name	Status
DatProgrammer	Online
DisProgrammer	Online

Events

Timestamp	Event Type
4/25/2023 11:25:45 AM	HandlerOnlineStatusChanged
4/25/2023 11:25:26 AM	ProgrammerConnected

The system tile helps you effectively monitor and manage your system by offering a clear view of your system's status, type, and adoption status at the top of each tile.

Overview tab

On the **Overview** tab, you can find more details about your system configurations and installed software. For example, expanding the **System Information** panel displays the following information:

Attribute	Description
Hostname	The system's assigned hostname.
Known IP Address(es)	The IP address(es) assigned to the system.
Operating System	The system's operating system.
Machine Manager Version	The version of the Machine Manager software.
Machine Manager Identifier	The unique identifier for this system.
Factory Name	The name of the factory where the system is used.

While expanding the **Installed Software** panel displays a list of installed software including name, version, and installation directory.

The **Programmers** section allows you to easily add, discover, and manage programmers connected to the system. For information about configuring programmers, see Configuration.

The **Events** section at bottom keeps track of system events in a table to simplify system monitoring activities.

Column	Description
Timestamp	The date and time when the event occurred.
Event Type	The category of the event
Message	A brief description of the event.

Click a row in the table to open a more detailed view of the event message (ex. additional information about the event and any related actions).

Configuration tab

The **Configuration** tab lets you set options for the following (as described in the Configuration page):

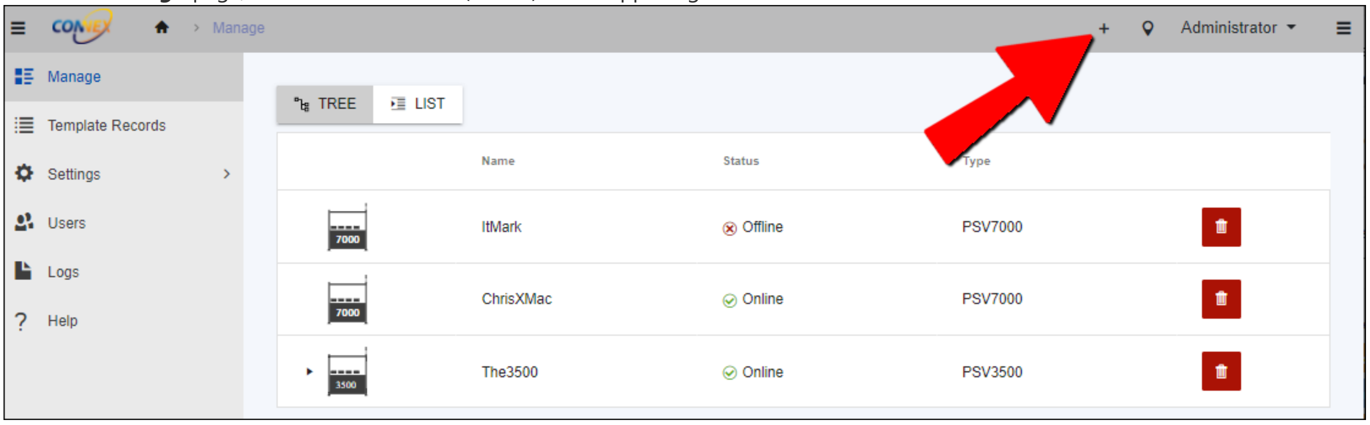
- LumenX options (Programmer Discovery Network Interface)
- FlashCORE III options (FC3 communication port and Programmer Discovery Network Interface)
- Machine Manager options (Machine Name, Factory Name, and Handler Type)
- MQTT service options (Machine Manager MQTT port number)

Logs tab

The **Logs** tab allows you to remotely view and download Machine Manager logs for efficient troubleshooting and issue resolution. Download the logs as a ZIP file to a directory of your choice, then review them on your local machine for in-depth analysis.

Adding a New System

From the **Manage** page, click Add Connection (+ icon) in the upper-right corner.



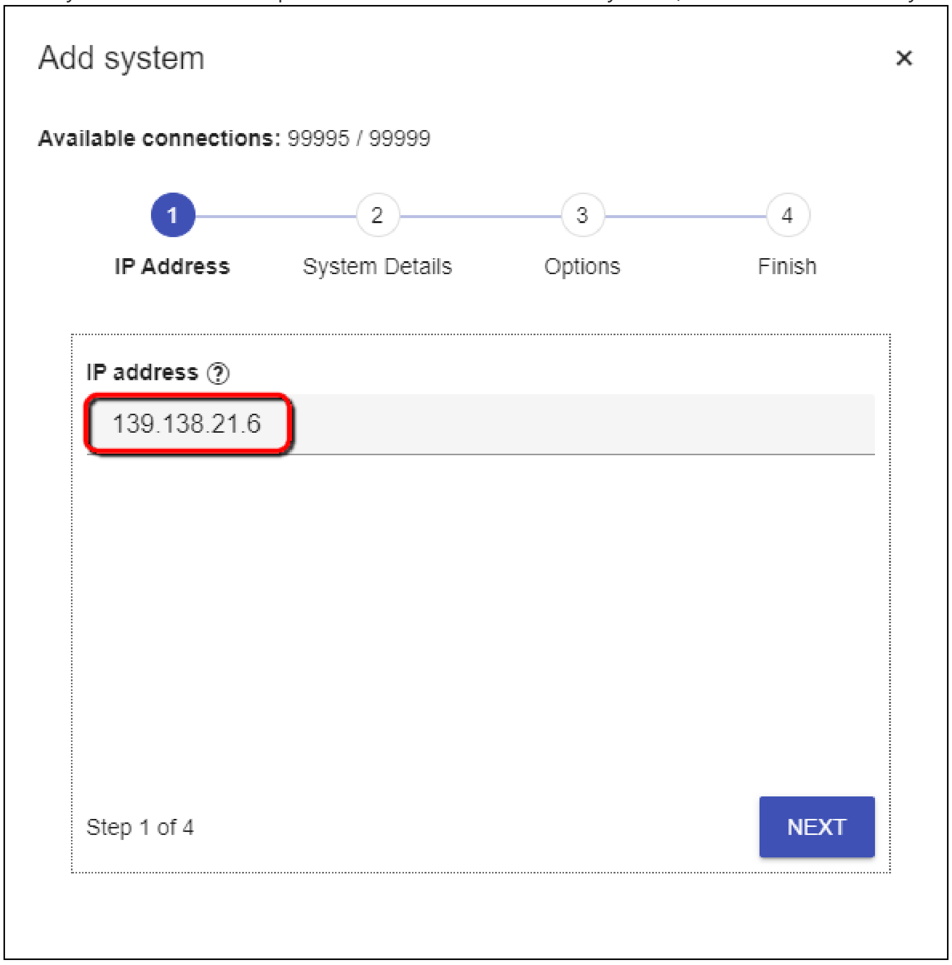
	Name	Status	Type
7000	ItMark	Offline	PSV7000
7000	ChrisXMac	Online	PSV7000
3500	The3500	Online	PSV3500

Note
The **Available connections** count on the **Add system** page indicates how many Machine Manager connections you can add with the current license. Confirm you have available connections to proceed.

Step 1: IP Address

Enter the IP address of the PSV system in the provided field. If you're unsure of the IP address, here's how you can find it on a Windows computer:

1. On the PSV system (Machine Manager instance) to add to ConneX Server, open the Command Prompt by typing **cmd** in the Windows search bar, then select **Command Prompt** from the results.
2. In the Command Prompt window, type the command **ipconfig**, and press **Enter**.
3. Look for the **IPv4 Address** entry for the network adapter used to connect to the PSV system (which is the IP address you enter into the **Add system** wizard).


Add system

Available connections: 99995 / 99999

IP address: 139.138.21.6

Step 2: System Details

If the connection to the PSV system is successful, the wizard displays a table with details of the new system. Confirm the table shows the intended system to add; else click **Back** to confirm the IP address.

Detail	Description
System Name	The ConneX given name of the system.
Machine Serial Number	The serial number of the system.
Hostname	The name of the host used by Windows.
System Type	The type of system (e.g. PSV7000, PSV5000, etc.)

Step 3: Options

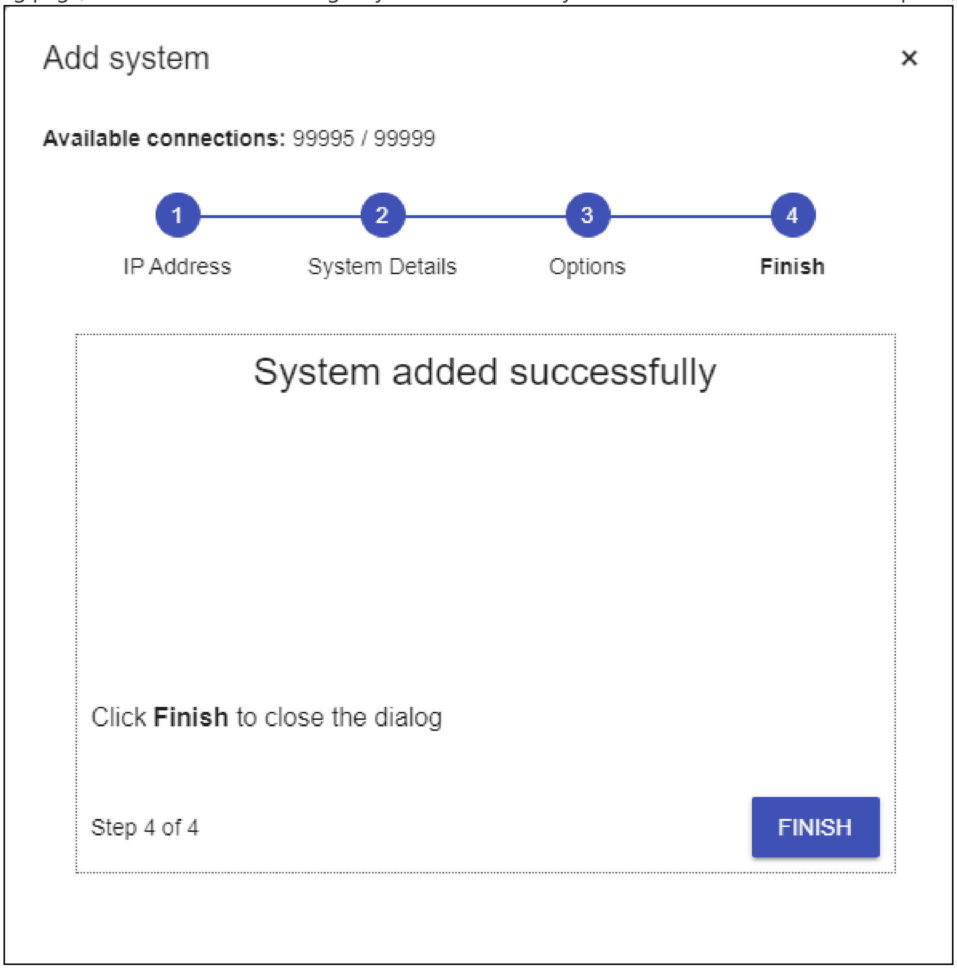
Set additional options to customize the connection process.

Option	Description

Option	Description
Adopt system after connecting?	Enable to adopt the system and allow ConneX Server to start receiving messages immediately; Disable to simply add the system now (ex. ConneX data transfer to be enabled later)
System Name	Type the name of the system.

Step 4: Finish

If the setup wizard displays the following page, then the Machine Manager system is successfully added to ConneX Server. At this point, you can click **Finish** to exit the wizard.


Add system

Available connections: 99995 / 99999

System added successfully

Click **Finish** to close the dialog

Step 4 of 4

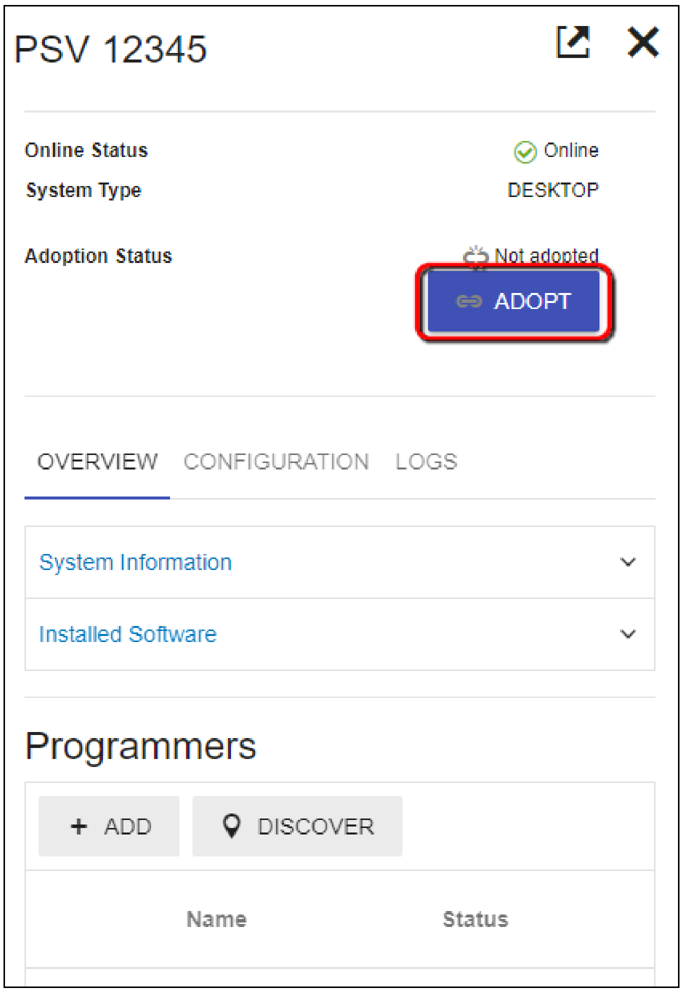
However, in some cases, a warning message might appear indicating that the adoption was not successful because the system is already adopted. In such cases, please follow the troubleshooting steps for System is adopted by another ConneX Service

A notification in the lower-right corner also indicates the success or failure of the add system operation.

To ensure/enable ConneX data transfer for the added Machine Manager, you must Adopt it from the ConneX Server (if not already adopted):

1. From the **Manage**page, click the new Machine Manager/PSV System.

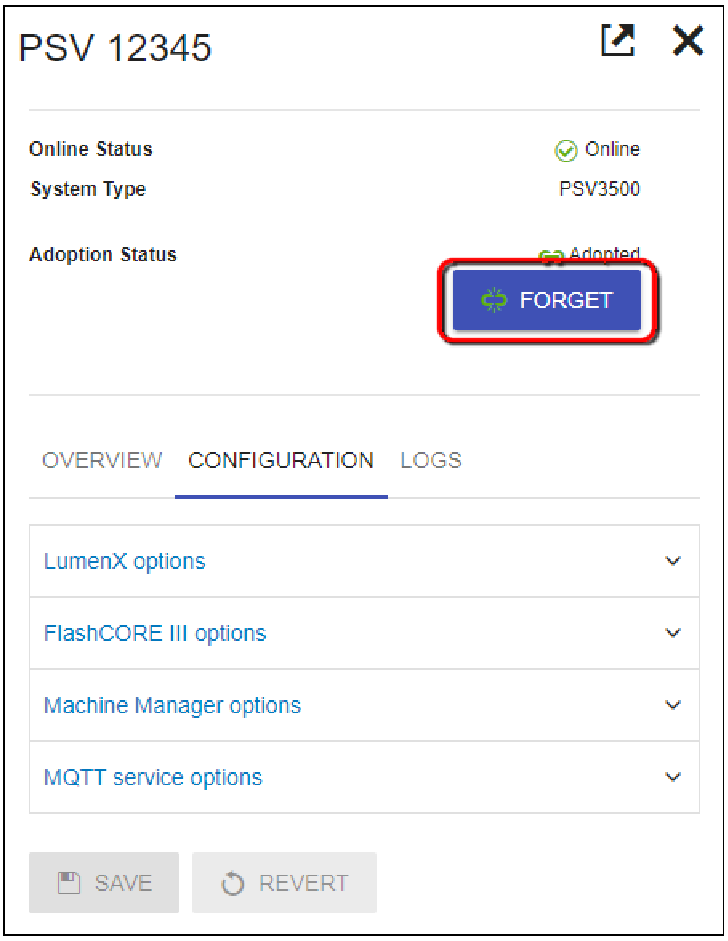
2. In the selected system tile, click **Adopt**.



Disable Machine Manager Data Transfer

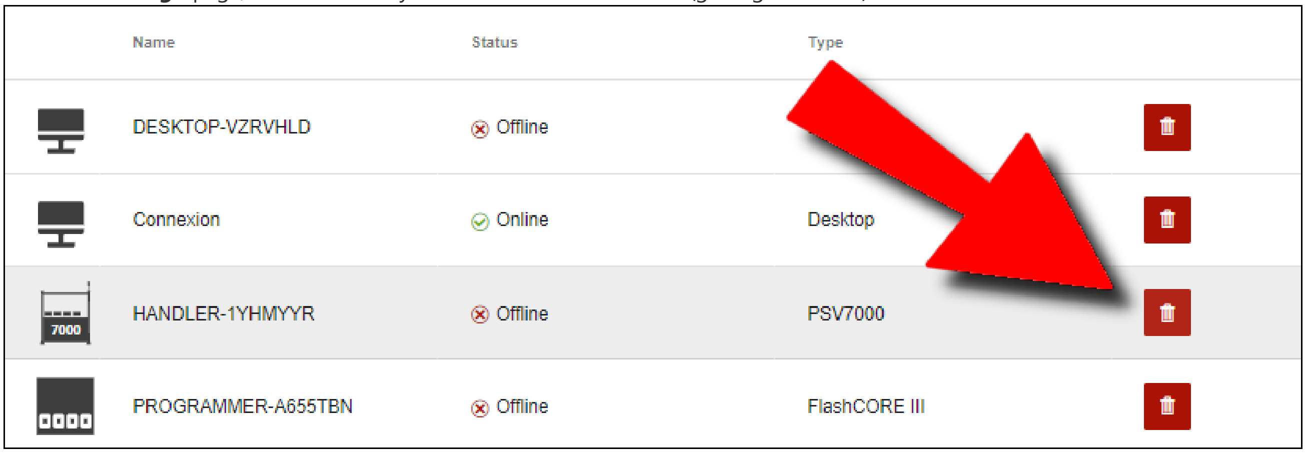
To disable ConneX data transfer, go to the **Manage** page and select/click the PSV System for which to stop/pause data transfer.

In the system tile, click **Forget**.



Removing a programming system/programmer

1. From the **Manage** page, locate the entity to remove and click **Delete** (garbage can icon).



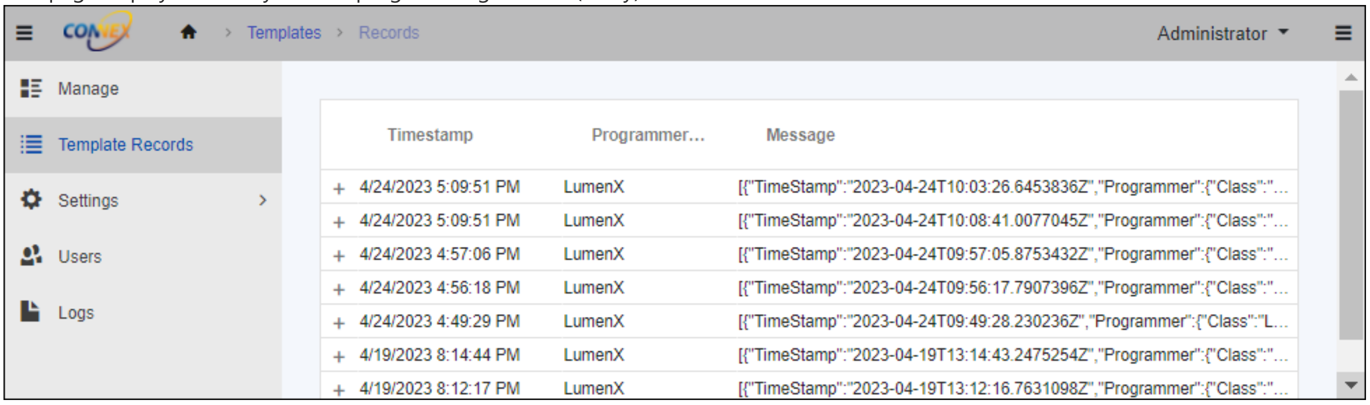
	Name	Status	Type	
	DESKTOP-VZRVHLD	Offline		
	Connexion	Online	Desktop	
	HANDLER-1YHMYYR	Offline	PSV7000	
	PROGRAMMER-A655TBN	Offline	FlashCORE III	

2. In the **Remove Entity** dialog, click **OK** to confirm the removal.

Caution
Removing an adopted programming system results in future records and events being lost. Proceed with caution.

Template Records

This page displays a list of your raw programming records (if any).



	Timestamp	Programmer...	Message
+	4/24/2023 5:09:51 PM	LumenX	[{"TimeStamp":"2023-04-24T10:03:26.6453836Z","Programmer":{"Class":"...
+	4/24/2023 5:09:51 PM	LumenX	[{"TimeStamp":"2023-04-24T10:08:41.0077045Z","Programmer":{"Class":"...
+	4/24/2023 4:57:06 PM	LumenX	[{"TimeStamp":"2023-04-24T09:57:05.8753432Z","Programmer":{"Class":"...
+	4/24/2023 4:56:18 PM	LumenX	[{"TimeStamp":"2023-04-24T09:56:17.7907396Z","Programmer":{"Class":"...
+	4/24/2023 4:49:29 PM	LumenX	[{"TimeStamp":"2023-04-24T09:49:28.230236Z","Programmer":{"Class":"L...
+	4/19/2023 8:14:44 PM	LumenX	[{"TimeStamp":"2023-04-19T13:14:43.2475254Z","Programmer":{"Class":"...
+	4/19/2023 8:12:17 PM	LumenX	[{"TimeStamp":"2023-04-19T13:12:16.7631098Z","Programmer":{"Class":"...

For more information, see the "Template Records" section on the Settings.

Settings

The **Settings** page displays a list of configuration parameters for ConneX.

Licensing

This section shows the software licensing information for ConneX service, including:

- License type (ex. Trial or Perpetual for ConneX Service)
- Available Connections (remaining number of client connections, or max connections minus used connections; a client connection is an instance of the Machine Manager Service connecting to a centralized ConneX Service)
- Maximum Connections (number of client connections licensed; possible number of Machine Manager Service connections to ConneX Service)
- ConneX Annual Maintenance Contract (expiration date for Machine Manager connections to ConneX Service)

To export or download a copy of the ConneX data for SentriX jobs, this section also provides a **Download SentriX Report** button.

For more information about your Data I/O software product licenses (including ConneX):

1. On the Handler/Host PC (of your PSV System), open a browser window (Chrome 100.0.x recommended) to the Data I/O License Manager at http://localhost:5002/.
2. On the **License Information** page, under the **ConneX** section, click **ConneX Service** to expand and view its license details.

Users Settings

The **Users Settings** section shows the password options for user accounts:

Field	Description
Password requires digit	User password requires at least one digit
Password requires lowercase	User password requires at least one lowercase character
Password requires uppercase	User password requires at least one uppercase character
Password requires non alphanumeric	User password requires at least one non-alphanumeric character
Password minimum length	Type the minimum number of characters required for user passwords

MQTT Connections

The **MQTT Connections** section shows the configuration for MQTT ports:

Field	Description
MQTT Broker Port	The TCP port number for the ConneX MQTT Broker (default TCP port is 1883)
MQTT Machine Manager Client Port	The TCP port number for the ConneX MQTT Machine Manager Client (default TCP port is 9002)

Template Records

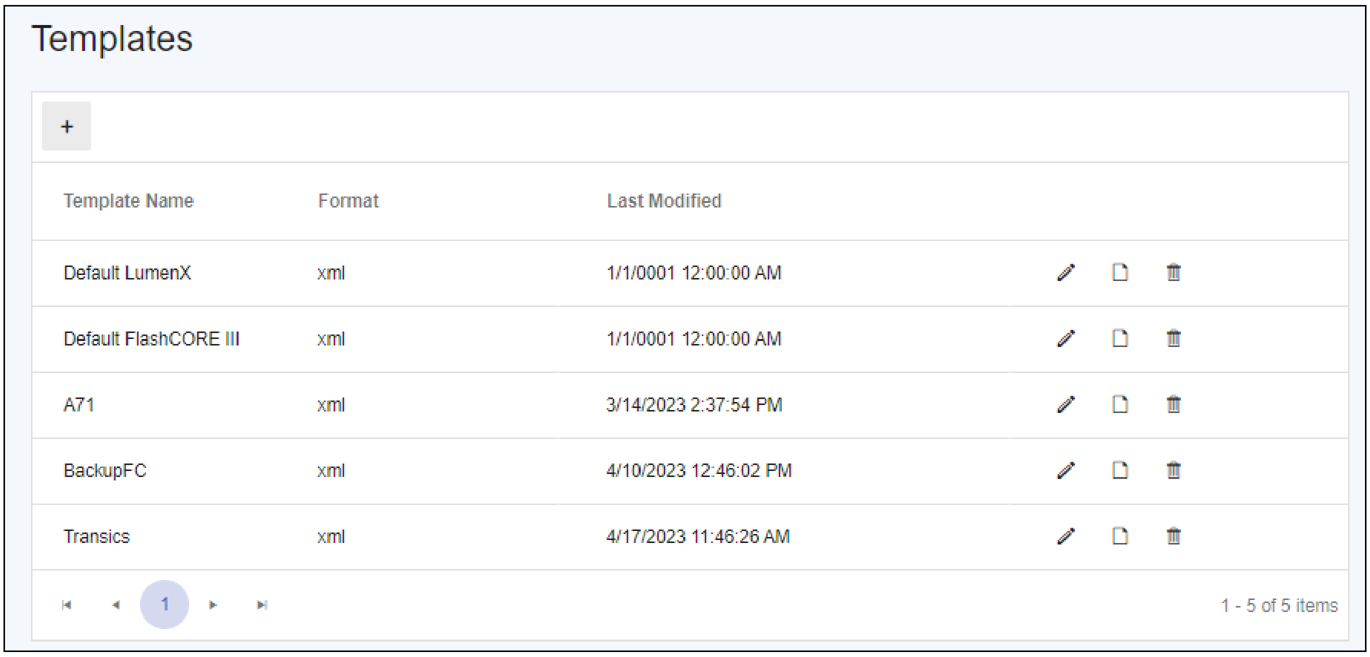
In addition to accessing ConneX data through MQTT and GraphQL queries, the latest ConneX version also allows users of previous versions to access ConneX data through standard templates for backwards-compatibility. The Template Records component and service publishes ConneX data through templates for legacy users of Template Manager in previous versions.

First, you create a custom template (such as .XML and .JSON) to define the specific data fields to collect. Then as devices are programmed, ConneX writes the programming statistics to the template-based record to effectively capture the user-specified data points.

By publishing the specified fields data (ex. programming statistics) through standard templates, ConneX simplifies the process of integrating your programming data into your Manufacturing Execution System (MES) or other data processing application(s).

The **Template Records** page displays a list of your raw programming records while any customized template records are stored in **C:\ProgramData\DataIO\ConneX\TemplateManager\Audit\Output**. To modify template settings, click **Settings** (in the left navigation pane), then click **Template Manager**.

The **Templates** section lists your current templates and allows you to add a new template (using the Plus button at top), delete an existing template (using the corresponding Trash button in right column), or specify a different file for an existing template (using Pencil button in right column).



Templates

+

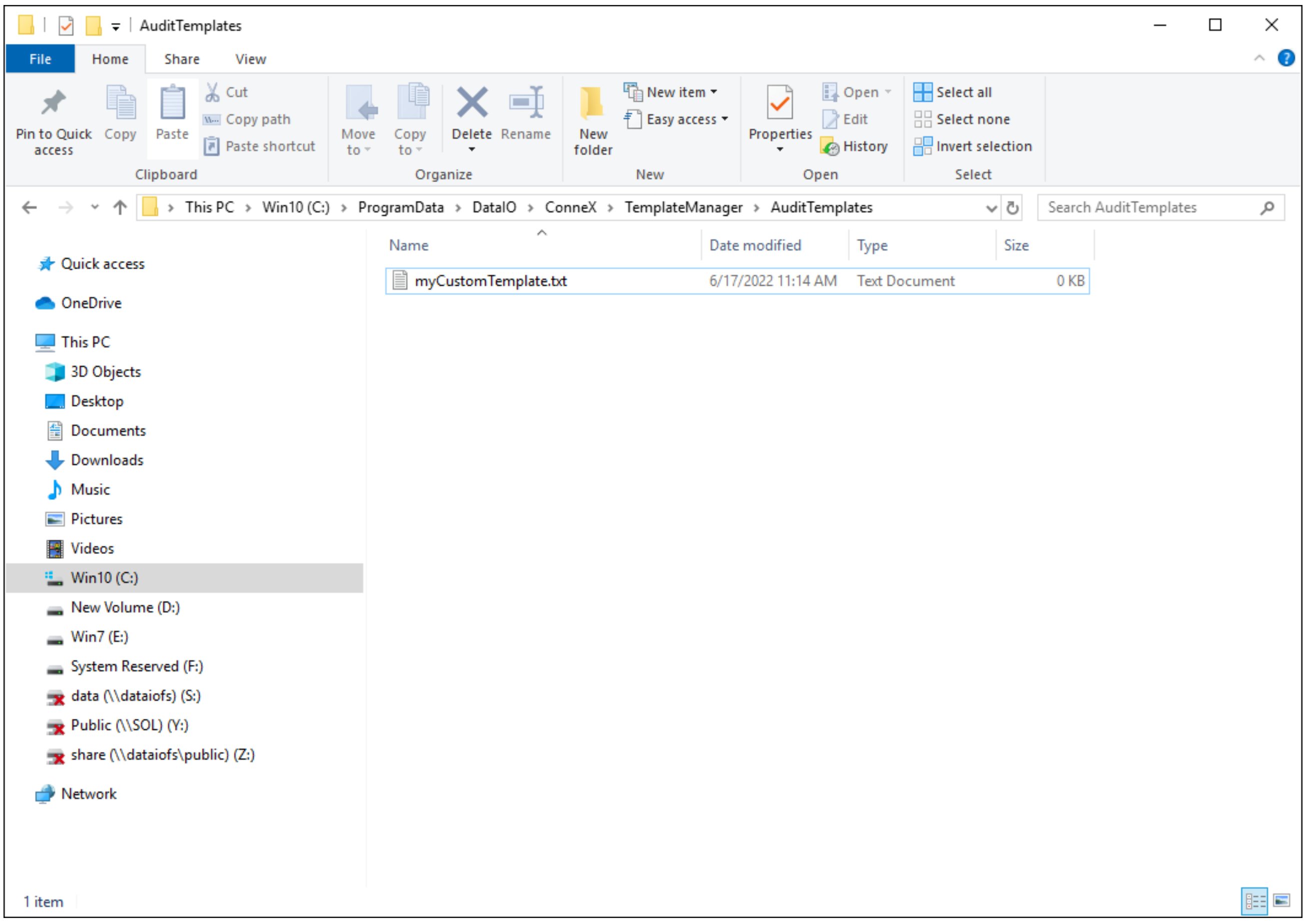
Template Name	Format	Last Modified
Default LumenX	xml	1/1/0001 12:00:00 AM
Default FlashCORE III	xml	1/1/0001 12:00:00 AM
A71	xml	3/14/2023 2:37:54 PM
BackupFC	xml	4/10/2023 12:46:02 PM
Transics	xml	4/17/2023 11:46:26 AM

1 - 5 of 5 items

Customize a Default Template

Complete the following steps to create a custom template (using .XML as an example) using the fields from a default template.

1. Start Windows/File Explorer and navigate to **C:\ProgramData\DataIO\ConneX\TemplateManager\AuditTemplates**.
2. Right-click anywhere in the right pane, point to **New**, click **Text Document**, type a name for the custom template, and then press ENTER.

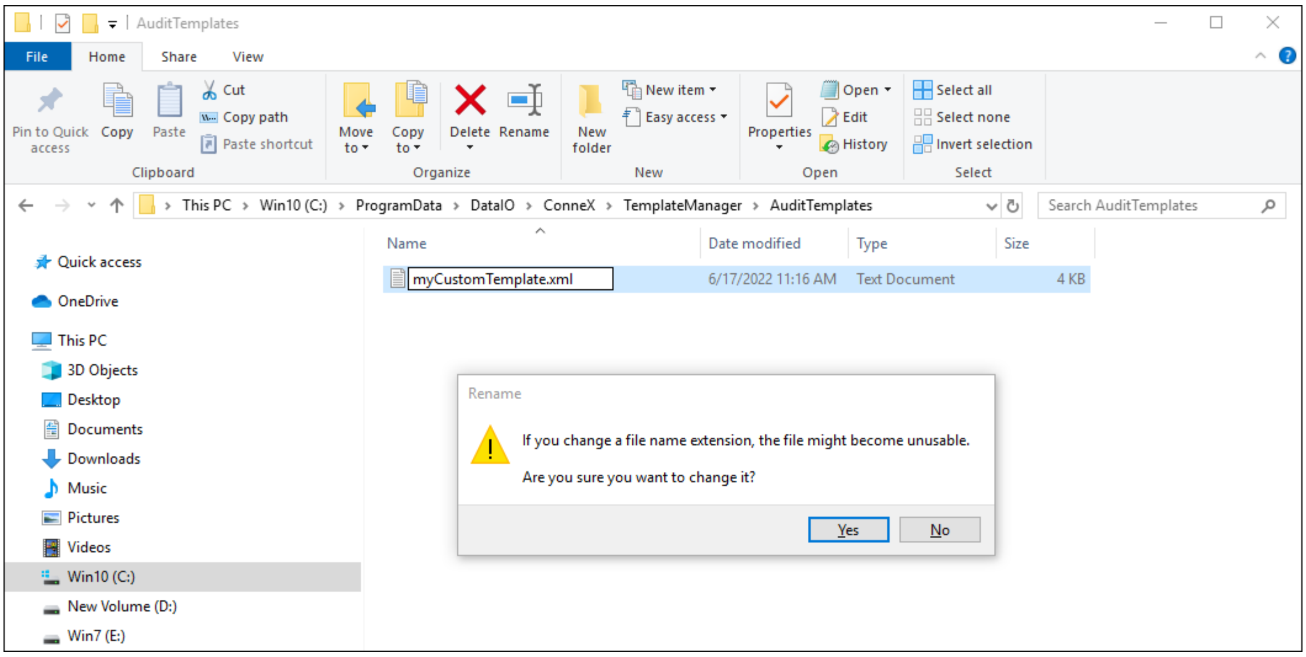


This PC > Win10 (C:) > ProgramData > DataIO > ConneX > TemplateManager > AuditTemplates

Name	Date modified	Type	Size
myCustomTemplate.txt	6/17/2022 11:14 AM	Text Document	0 KB

1 item

3. Open a default template and **copy** all of its fields.
4. Now open your custom template file and **paste** the default fields into your custom template, then **Save** your template.
5. Modify the fields in your custom template as desired (ex. add, remove, reorder fields), then **Save**.
6. Finally, rename the custom template file from **.TXT** to **.XML**.



Name	Date modified	Type	Size
myCustomTemplate.xml	6/17/2022 11:16 AM	Text Document	4 KB

Rename

If you change a file name extension, the file might become unusable.

Are you sure you want to change it?

Add Custom Template to ConneX

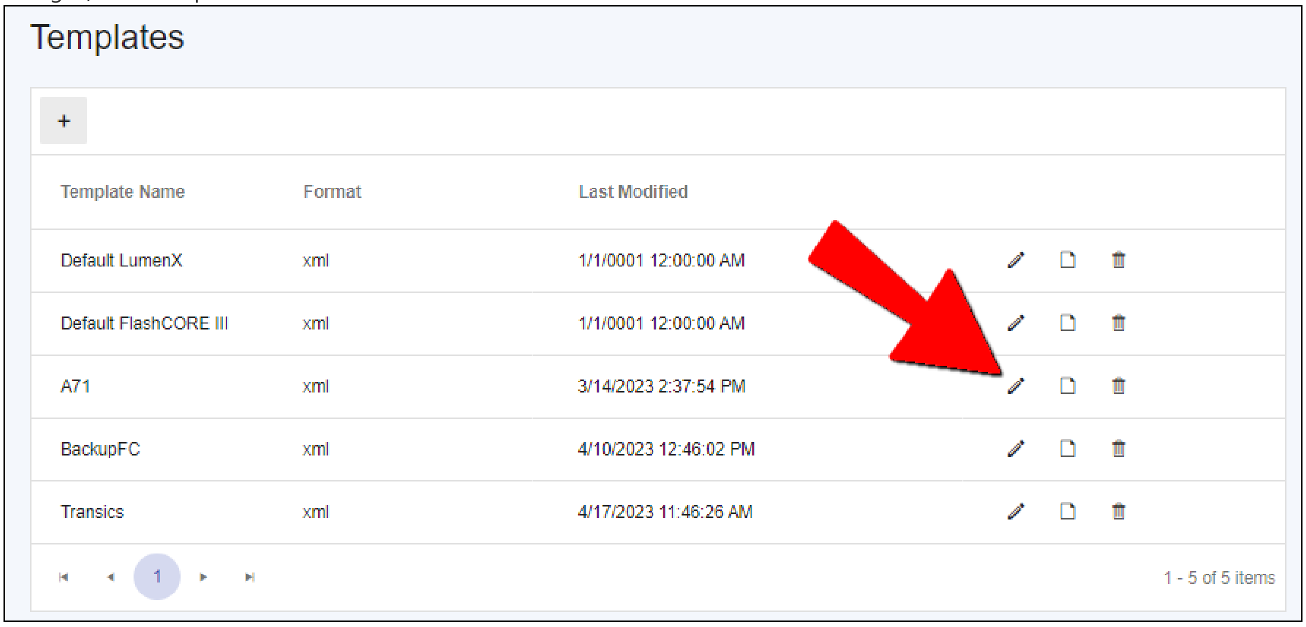
Complete the following steps to add your custom template (using .XML as an example) to ConneX.

1. Open a browser window to http://localhost:5001 and login.
2. In the left pane, expand **Settings**, and then click **Template Manager**.
3. On the **Templates** page, click the add template (**+**) button near the upper-left corner.
4. Type a **Template Name**, and then click **Select File**.
5. In the **Open** dialog box, browse to and select your custom template, then click **Open**.
6. Click **Save**.

Set ConneX to Use Added Custom Template

Complete the following steps to set your custom template as the format for ConneX to generate and output its programming statistics.

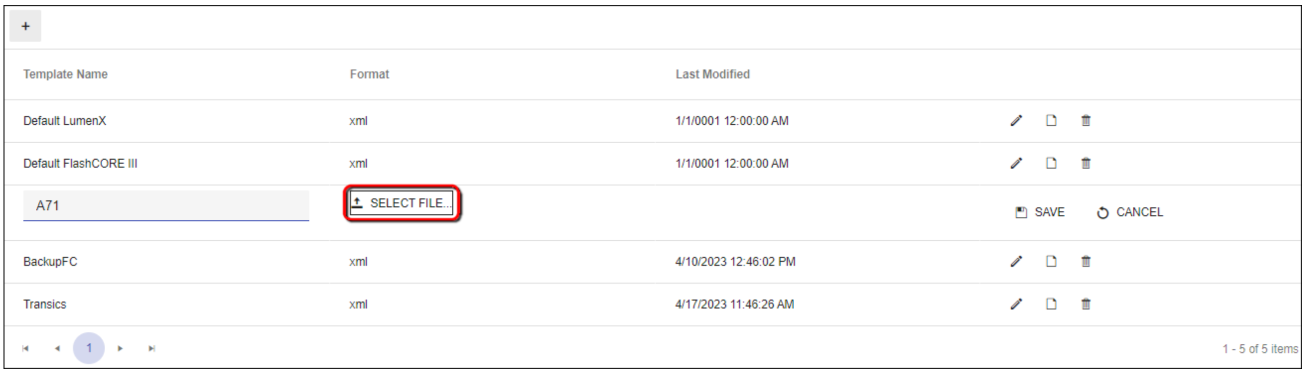
1. In the **Templates** table, click the Pencil icon/button for the specific template that you want to change (default templates cannot be changed). For example:



Templates

Template Name	Format	Last Modified
Default LumenX	xml	1/1/0001 12:00:00 AM
Default FlashCORE III	xml	1/1/0001 12:00:00 AM
A71	xml	3/14/2023 2:37:54 PM
BackupFC	xml	4/10/2023 12:46:02 PM
Transics	xml	4/17/2023 11:46:26 AM

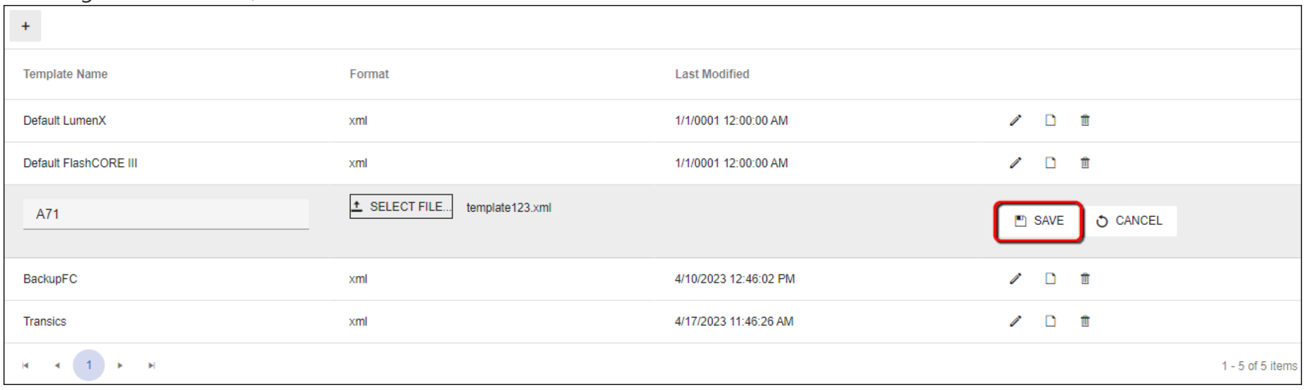
2. Click the **Select File** button, browse and select your new custom template, and then click **Open**.



Template Name	Format	Last Modified	
Default LumenX	xml	1/1/0001 12:00:00 AM	
Default FlashCORE III	xml	1/1/0001 12:00:00 AM	
A71	SELECT FILE...		SAVE CANCEL
BackupFC	xml	4/10/2023 12:46:02 PM	
Transics	xml	4/17/2023 11:46:26 AM	

1 - 5 of 5 items

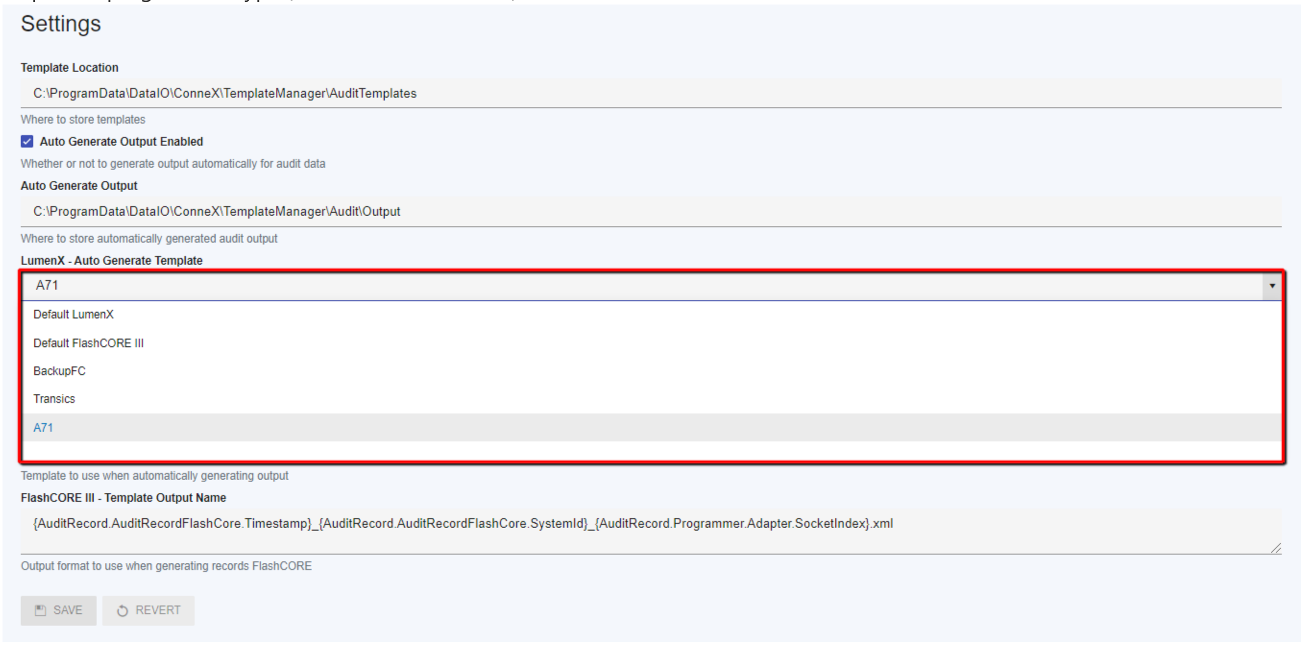
3. In the right-most column, click **Save**.



Template Name	Format	Last Modified	
Default LumenX	xml	1/1/0001 12:00:00 AM	
Default FlashCORE III	xml	1/1/0001 12:00:00 AM	
A71	SELECT FILE... template123.xml		SAVE CANCEL
BackupFC	xml	4/10/2023 12:46:02 PM	
Transics	xml	4/17/2023 11:46:26 AM	

1 - 5 of 5 items

4. Scroll down to the **Settings** section and from the **Auto Generate Template** drop-down list, select the custom template for the respective programmer type (LumenX or FlashCORE).



Settings

Template Location
C:\ProgramData\DataIO\ConneX\TemplateManager\AuditTemplates
Where to store templates

☑ Auto Generate Output Enabled
Whether or not to generate output automatically for audit data

Auto Generate Output
C:\ProgramData\DataIO\ConneX\TemplateManager\Audit\Output
Where to store automatically generated audit output

LumenX - Auto Generate Template
A71
- Default LumenX
- Default FlashCORE III
- BackupFC
- Transics
- A71

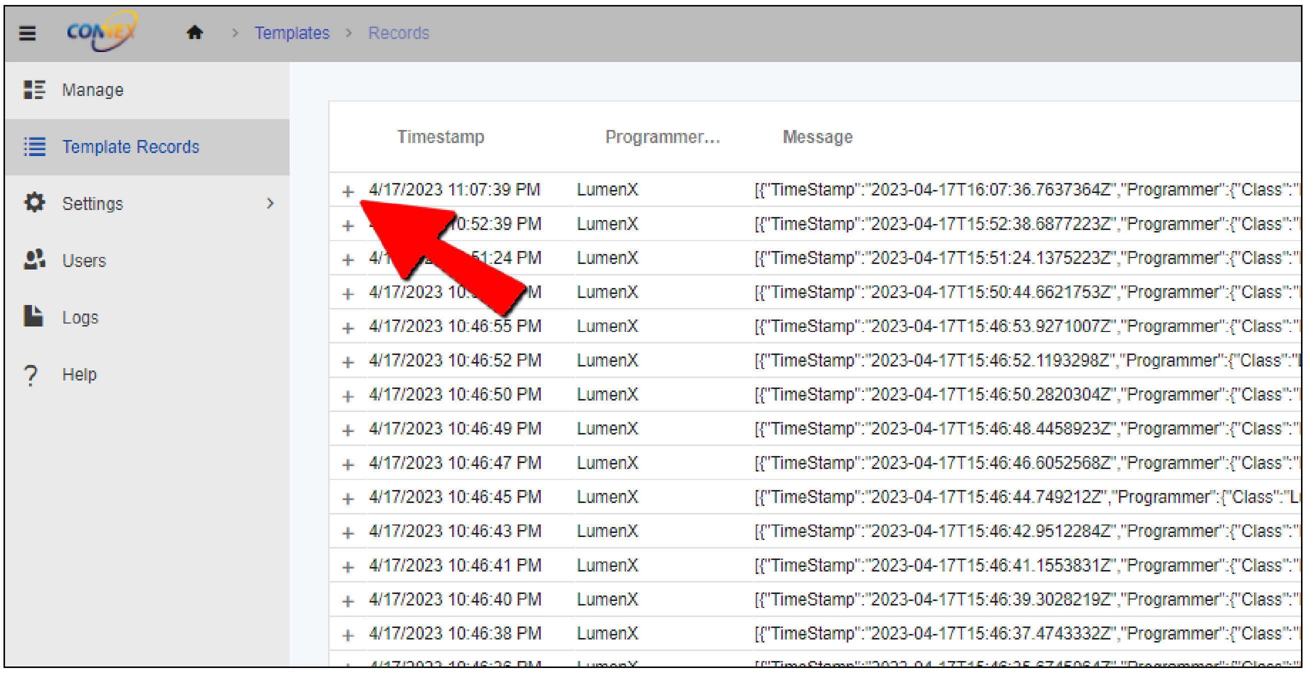
Template to use when automatically generating output

FlashCORE III - Template Output Name
{AuditRecord.AuditRecordFlashCore.Timestamp}_{AuditRecord.AuditRecordFlashCore.SystemId}_{AuditRecord.Programmer.Adapter.SocketIndex}.xml
Output format to use when generating records FlashCORE

Test Output to Custom Template

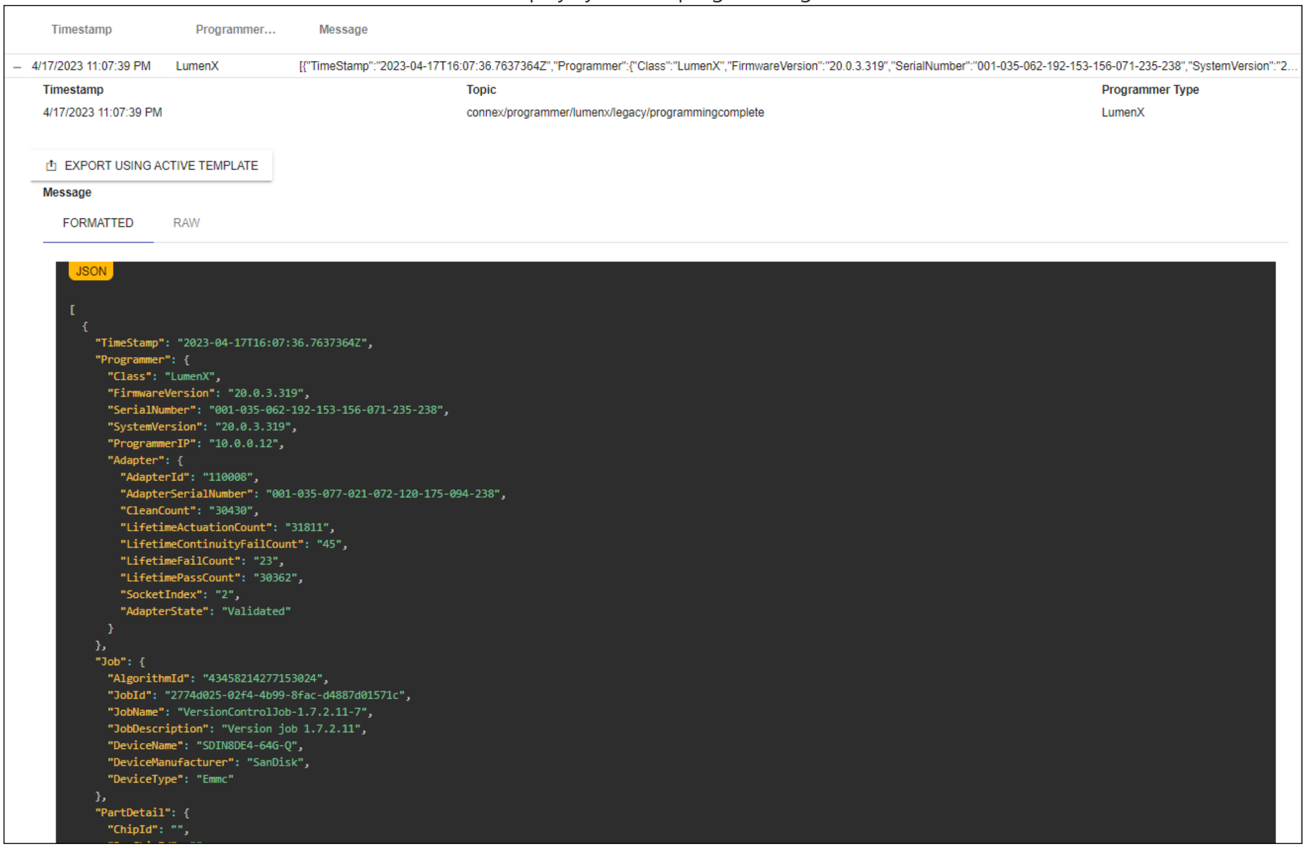
Before running a device programming job in production, first complete the following steps to test and ensure that your custom template is working (that ConneX writes its programming data using the custom template you created and specified).

1. Run a job with **Pass Limit** of one or two devices.
2. After the job completes, return to http://localhost:5001, and click **Template Records**.
3. In the right pane, click the plus (**+**) button to expand the audit record.



	Timestamp	Programmer…	Message
+	4/17/2023 11:07:39 PM	LumenX	[{"TimeStamp":"2023-04-17T16:07:36.7637364Z","Programmer":{"Class":"
+	0:52:39 PM	LumenX	[{"TimeStamp":"2023-04-17T15:52:38.6877223Z","Programmer":{"Class":"
+	4/1 1:24 PM	LumenX	[{"TimeStamp":"2023-04-17T15:51:24.1375223Z","Programmer":{"Class":"
+	4/17/2023 10: M	LumenX	[{"TimeStamp":"2023-04-17T15:50:44.6621753Z","Programmer":{"Class":"
+	4/17/2023 10:46:55 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:53.9271007Z","Programmer":{"Class":"
+	4/17/2023 10:46:52 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:52.1193298Z","Programmer":{"Class":"
+	4/17/2023 10:46:50 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:50.2820304Z","Programmer":{"Class":"
+	4/17/2023 10:46:49 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:48.4458923Z","Programmer":{"Class":"
+	4/17/2023 10:46:47 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:46.6052568Z","Programmer":{"Class":"
+	4/17/2023 10:46:45 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:44.749212Z","Programmer":{"Class":"L
+	4/17/2023 10:46:43 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:42.9512284Z","Programmer":{"Class":"
+	4/17/2023 10:46:41 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:41.1553831Z","Programmer":{"Class":"
+	4/17/2023 10:46:40 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:39.3028219Z","Programmer":{"Class":"
+	4/17/2023 10:46:38 PM	LumenX	[{"TimeStamp":"2023-04-17T15:46:37.4743332Z","Programmer":{"Class":"

4. Now scroll down to view and confirm that ConneX displays your raw programming records data.

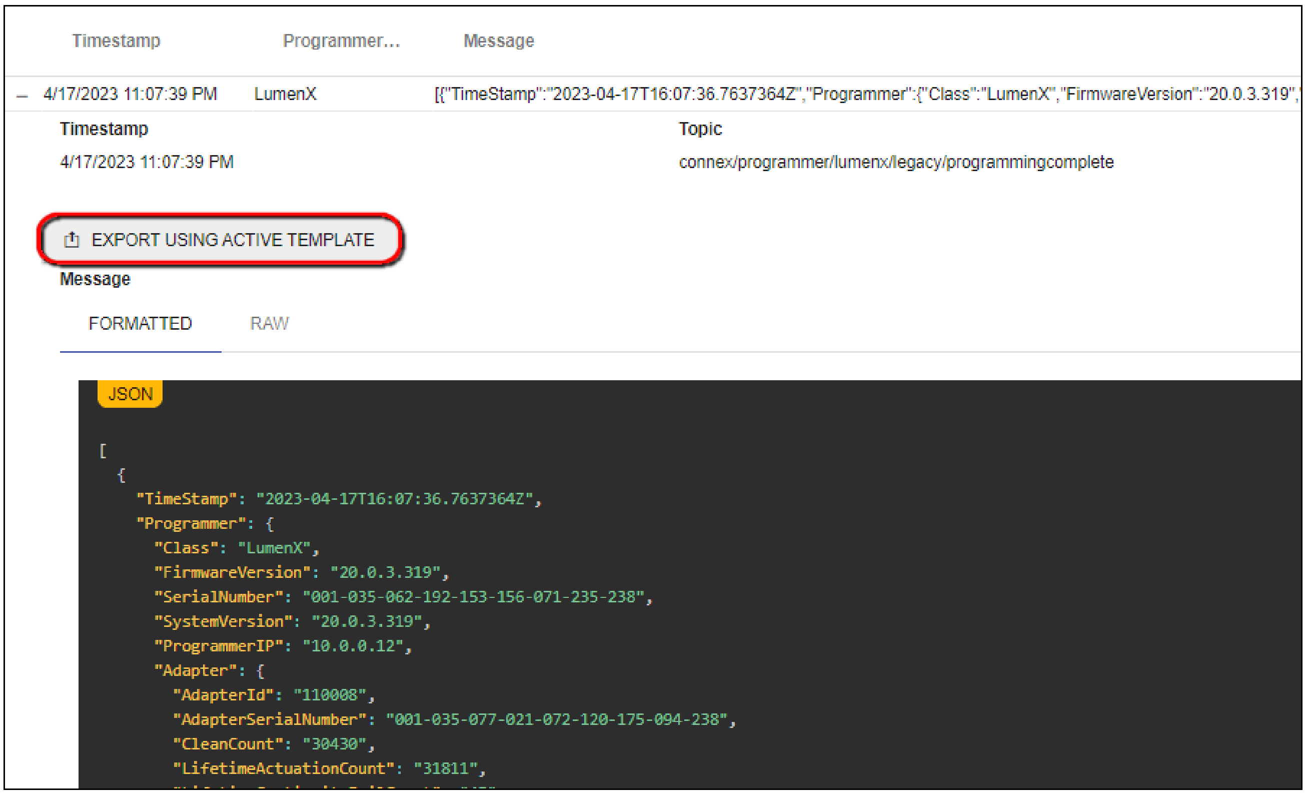


	Timestamp	Programmer…	Message
–	4/17/2023 11:07:39 PM	LumenX	[{"TimeStamp":"2023-04-17T16:07:36.7637364Z","Programmer":{"Class":"LumenX","FirmwareVersion":"20.0.3.319","SerialNumber":"001-035-062-192-153-156-071-235-238","SystemVersion":"2…

Timestamp	Topic	Programmer Type
4/17/2023 11:07:39 PM	connex/programmer/lumenx/legacy/programmingcomplete	LumenX

```
[
  {
    "TimeStamp": "2023-04-17T16:07:36.7637364Z",
    "Programmer": {
      "Class": "LumenX",
      "FirmwareVersion": "20.0.3.319",
      "SerialNumber": "001-035-062-192-153-156-071-235-238",
      "SystemVersion": "20.0.3.319",
      "ProgrammerIP": "10.0.0.12",
      "Adapter": {
        "AdapterId": "110008",
        "AdapterSerialNumber": "001-035-077-021-072-120-175-094-238",
        "CleanCount": "30430",
        "LifetimeActuationCount": "31811",
        "LifetimeContinuityFailCount": "45",
        "LifetimeFailCount": "23",
        "LifetimePassCount": "30362",
        "SocketIndex": "2",
        "AdapterState": "Validated"
      }
    },
    "Job": {
      "AlgorithmId": "434582142771153024",
      "JobId": "2774d025-02f4-4b99-8fac-d4887d01571c",
      "JobName": "VersionControlJob-1.7.2.11-7",
      "JobDescription": "Version job 1.7.2.11",
      "DeviceName": "SDIN8DE4-64G-Q",
      "DeviceManufacturer": "SanDisk",
      "DeviceType": "Emmc"
    },
    "PartDetail": {
      "ChipId": "",
```

5. To view the same programming statistics through a customized template: a. Return to the **Settings** page, select a different template (from the **Auto Generate Template** drop-down list) and **Save**. b. Now on the **Template Records** page, select the same record, and click **Export Using Active Template**.



– 4/17/2023 11:07:39 PM LumenX [{"TimeStamp":"2023-04-17T16:07:36.7637364Z","Programmer":{"Class":"LumenX","FirmwareVersion":"20.0.3.319",

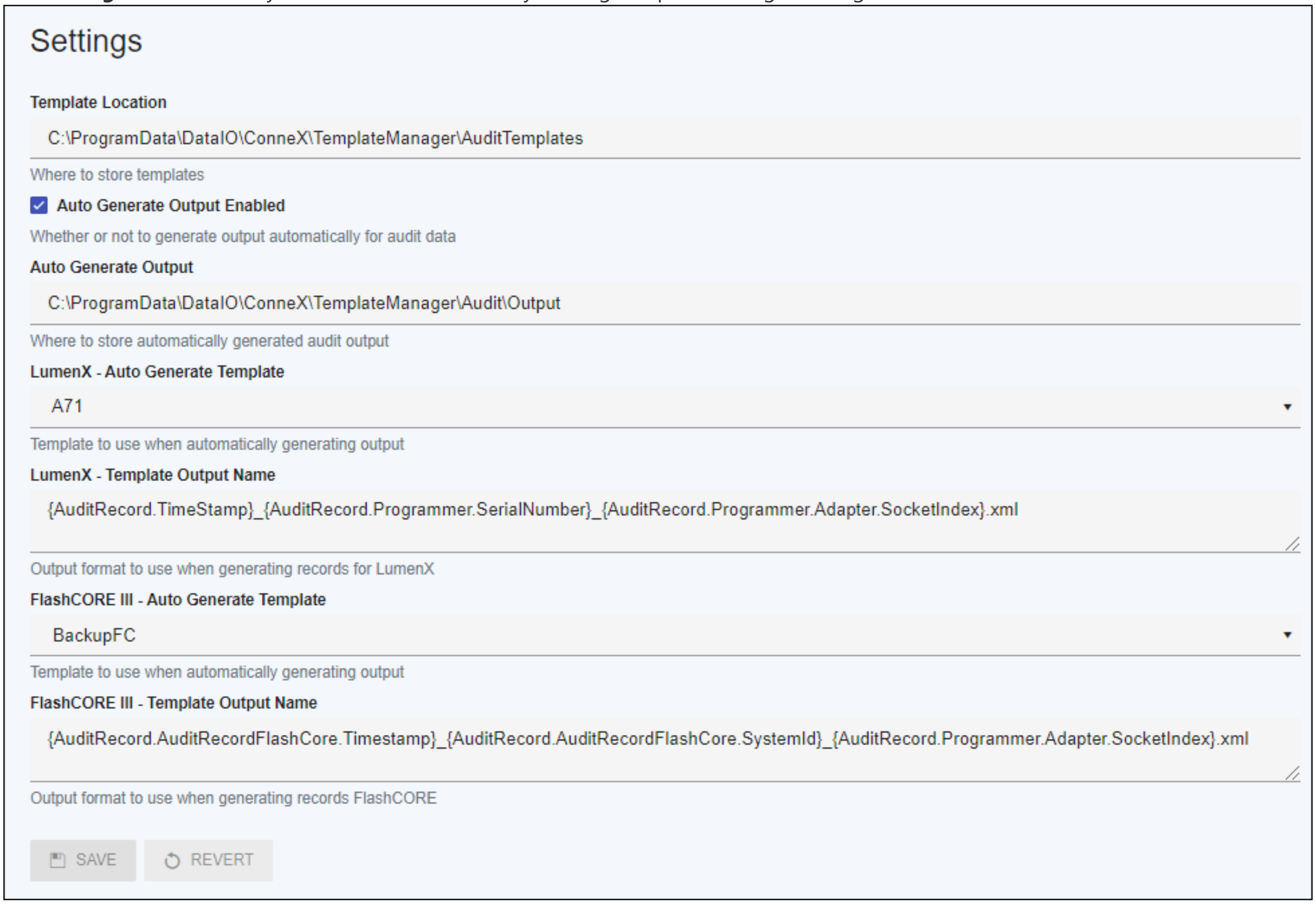
Timestamp
4/17/2023 11:07:39 PM

Topic
connex/programmer/lumenx/legacy/programmingcomplete

```
[
  {
    "TimeStamp": "2023-04-17T16:07:36.7637364Z",
    "Programmer": {
      "Class": "LumenX",
      "FirmwareVersion": "20.0.3.319",
      "SerialNumber": "001-035-062-192-153-156-071-235-238",
      "SystemVersion": "20.0.3.319",
      "ProgrammerIP": "10.0.0.12",
      "Adapter": {
        "AdapterId": "110008",
        "AdapterSerialNumber": "001-035-077-021-072-120-175-094-238",
        "CleanCount": "30430",
        "LifetimeActuationCount": "31811",
```

c. Repeat Step 3 above to view the same data in the new template (or go to **C:\ProgramData\DataIO\ConneX\TemplateManager\Audit\Output**).

The **Settings** section allows you to review and/or modify existing Template Manager settings.


Settings

Template Location
C:\ProgramData\DataIO\ConneX\TemplateManager\AuditTemplates
Where to store templates

☑ Auto Generate Output Enabled
Whether or not to generate output automatically for audit data

Auto Generate Output
C:\ProgramData\DataIO\ConneX\TemplateManager\Audit\Output
Where to store automatically generated audit output

LumenX - Auto Generate Template
A71
Template to use when automatically generating output

LumenX - Template Output Name
{AuditRecord.TimeStamp}_{AuditRecord.Programmer.SerialNumber}_{AuditRecord.Programmer.Adapter.SocketIndex}.xml
Output format to use when generating records for LumenX

FlashCORE III - Auto Generate Template
BackupFC
Template to use when automatically generating output

FlashCORE III - Template Output Name
{AuditRecord.AuditRecordFlashCore.Timestamp}_{AuditRecord.AuditRecordFlashCore.SystemId}_{AuditRecord.Programmer.Adapter.SocketIndex}.xml
Output format to use when generating records FlashCORE

Field	Description
Template Location	The directory containing your template files
Auto Generate Output Enabled	Check the box to enable automatic generation of audit output data; uncheck to disable
Auto Generate Output	The directory containing your automatically generated audit output data

Field	Description
LumenX Auto Generate Template	Specifies which template to use for LumenX jobs
LumenX Template Output Name	Specifies the output format for field records in the generated LumenX template
FlashCORE Ⅲ Auto Generate Template	Specifies which template to use for FlashCore jobs
FlashCORE Ⅲ TEmplate Output Name	Specifies the output format for field records in the generated FlashCORE template

Overview

ConneX provides a built-in user authentication system to prevent unauthorized access and modification of the system.

Roles

ConneX has the following roles that users can be assigned:

Roles	Description
Admin	Unrestricted access to the ConneX portal.
User	Read-only access to most pages.

Manage

Click here to be redirected to the user management page. There you will find a list of users for this instance of ConneX.

Adding a new user ADMIN

1. From the Users page, click the **Add User** button in the upper right.
2. Set the username.
3. (optional) Enter first and last name.
4. Enter a password that conforms to the settings found here.
5. Choose a role from the available list of roles.
6. Click the **Save** button.

Editing a user ADMIN

1. From the Users page, find the user in the table and click the **Edit User** button.
2. (optional) Enter first and last name.
3. (optional) Enter a new password that conforms to the settings found here.
4. Choose a role from the available list of roles.
5. Click the **Save** button.

Note
The **Administrator** user's role cannot be changed.

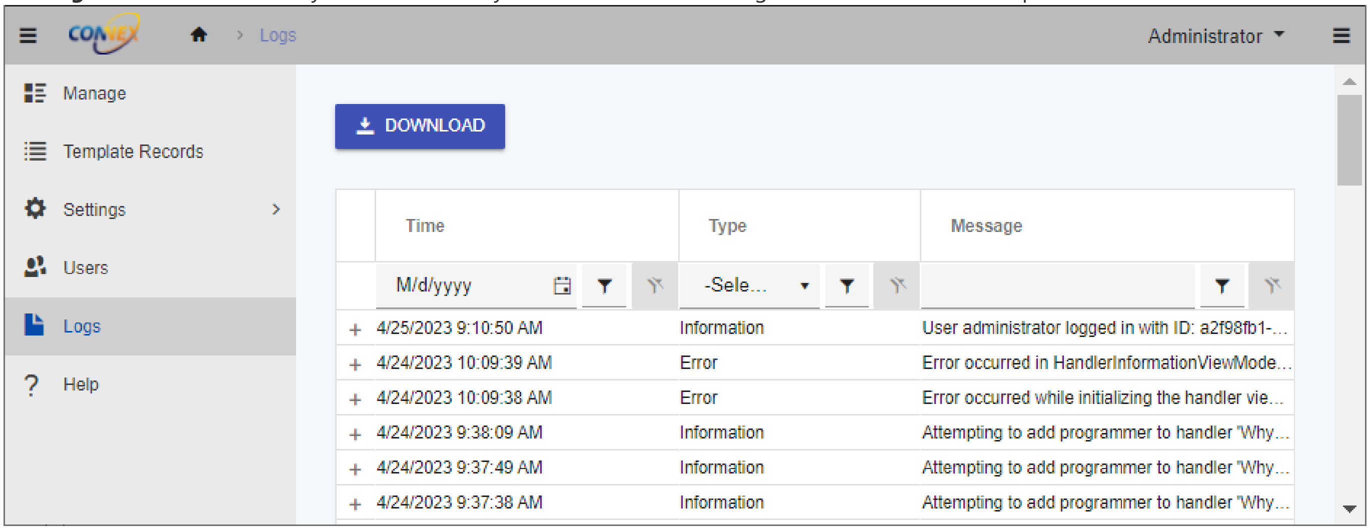
Deleting a user ADMIN

1. From the Users page, find the user in the table and click the **Delete User** button.
2. Confirm the removal by clicking the **Delete** button in the dialog.

Note
The **Administrator** user cannot be deleted.

Logs

The **Logs** node shows a list of system events that you can sort and filter using the column controls at top.



Time	Type	Message
4/25/2023 9:10:50 AM	Information	User administrator logged in with ID: a2f98fb1-…
4/24/2023 10:09:39 AM	Error	Error occurred in HandlerInformationViewMode…
4/24/2023 10:09:38 AM	Error	Error occurred while initializing the handler vie…
4/24/2023 9:38:09 AM	Information	Attempting to add programmer to handler 'Why…
4/24/2023 9:37:49 AM	Information	Attempting to add programmer to handler 'Why…
4/24/2023 9:37:38 AM	Information	Attempting to add programmer to handler 'Why…

To export a ZIP file of ConneX system log files, click the **Download** button, then choose a desired directory and filename.

Overview

The Troubleshooting section provides guidance for addressing common issues that may arise while using the software. This section aims to help users quickly identify and resolve problems that may impact their workflow, minimize downtime, and ensure that the software operates as intended.

The troubleshooting guidance in this section is organized into different categories that correspond to specific areas of the software. Each category contains a list of issues, along with a description of the symptoms and recommended solutions.

Before proceeding with any troubleshooting steps, ensure that you have reviewed the relevant documentation and have met all software requirements. In some cases, issues may be resolved simply by reviewing the documentation or performing basic troubleshooting steps.

If you are unable to resolve the issue using the guidance provided in this section, please contact technical support for additional assistance.

ConneX Service

Below you will find common issues found with the **ConneX Service**.

System is adopted by another ConneX Service

If you encounter an error indicating that a system has already been adopted by another **ConneX service**, there are two options to resolve this issue.

Option 1: "Forget" the System in the other instance of ConneX Service ADMIN

This option requires access to the other instance of **ConneX Service** that has already adopted the system. Follow these steps to "forget" the system in the other instance of **ConneX Service**:

1. Open the other instance of **ConneX Service** that has already adopted the system.
2. Select the system that you want to adopt in the desired **ConneX Service**.
3. Click the **Forget** button to remove the system from the previous **ConneX Service** instance.
4. Once the system has been forgotten, it can be adopted by the desired **ConneX Service**.

Option 2: "Force Adoption" as an Administrator ADMIN

This option allows the system to be adopted by the desired **ConneX Service** without requiring access to the other instance of **ConneX Service**. However, it is important to note that forcing adoption may result in the loss of data. Follow these steps to force the adoption of the system:

1. Open the **ConneX Service** on the system that you want to adopt in the desired **ConneX Service**.
2. Click the "Force Adoption" button.

Warning

Using the "Force adoption" option may result in the loss of data for the system, as any existing programming statistics for the system in the previous instance of **ConneX Service** will not be transferred. It is recommended to use this option only as a last resort and to ensure that all necessary backups are in place before proceeding.

No ConneX license found

If ConneX Server displays the "No ConneX license found" message), then complete the "Activate ConneX License" steps at the bottom of Installation. If ConneX service is running on a Windows Server 2016 computer, ensure you install Microsoft .NET Framework 4.7.2.

Change Log

This page lists all notable changes to ConneX.

[3.0.4] - (April, 2023)

New Features

- ConneX system installation is split into Machine Manager installer for PSV/Desktop systems and ConneX Software for ConneX host
- Add Handler wizard guides user through adding and adopting a handler.
- Tree View shows a hierarchical view, which makes it easier to see which programmers belong to which handler.
- Added support for PSV 3500 systems.

Enhancements/Bug Fixes

- Improvements to Help documentation.
- Template Manager record functionality fully integrated into ConneX service.
- SentriX reports show number of FlashCORE and LumenX devices programmed.
- Available fields in FlashCORE records fully populated.
- Handler and Programmer names can be changed for easier differentiation.
- Fixed an issue where switching adapters may cause a system to become unresponsive.
- ConneX Software now supports Windows Server 2016, 2019, and 2022.
- Improved performance for displaying programming records in ConneX.

[3.0.3] - (February 3, 2023)

New Features

- License Manager updated to 1.0.0.61.

Enhancements/Bug Fixes

- Increased querying performance of metrics.
- Updated licensing for Machine Manager connections to ConneX Service.
- Implemented demo/eval mode with restricted functionality (ex. Trial or Expired license, exceeded number of available connections...).
- Added UI notifications/indicators for Trial, Expired, and No license.

[3.0.0] - (July 6, 2022)

New Features

- License Manager updated to 1.0.0.60.
- Added support for MQTT API protocol for real time data.
- Added support for GraphQL API for querying historical data.
- Added UI to support configuration of system.

[2.0.1.133] - (March 15, 2021)

New Features

- Added support for larger log files to accommodate 32K serial numbers from LumenX DMS.
- Added support for larger templates.
- Updated License Manager to version 1.0.0.47 to match version in LumenX DMS.

Bug Fixes

- Improved SentriX billing performance.
- Removed support for double quotes in fields.

[2.0.0.87] - (March 19, 2019)

Bug Fixes

- Fixed audit record parsing stability.

[2.0.0.83] - (November 11, 2018)

Bug Fixes

- ConneX now unlocks and polls any FlashCORE programmers that were previously registered/locked to another machine.

API Overview

ConneX provides API hooks using the following standards:

- GraphQL
- MQTT

GraphQL

ConneX provides a GraphQL endpoint for querying data.

What is GraphQL?

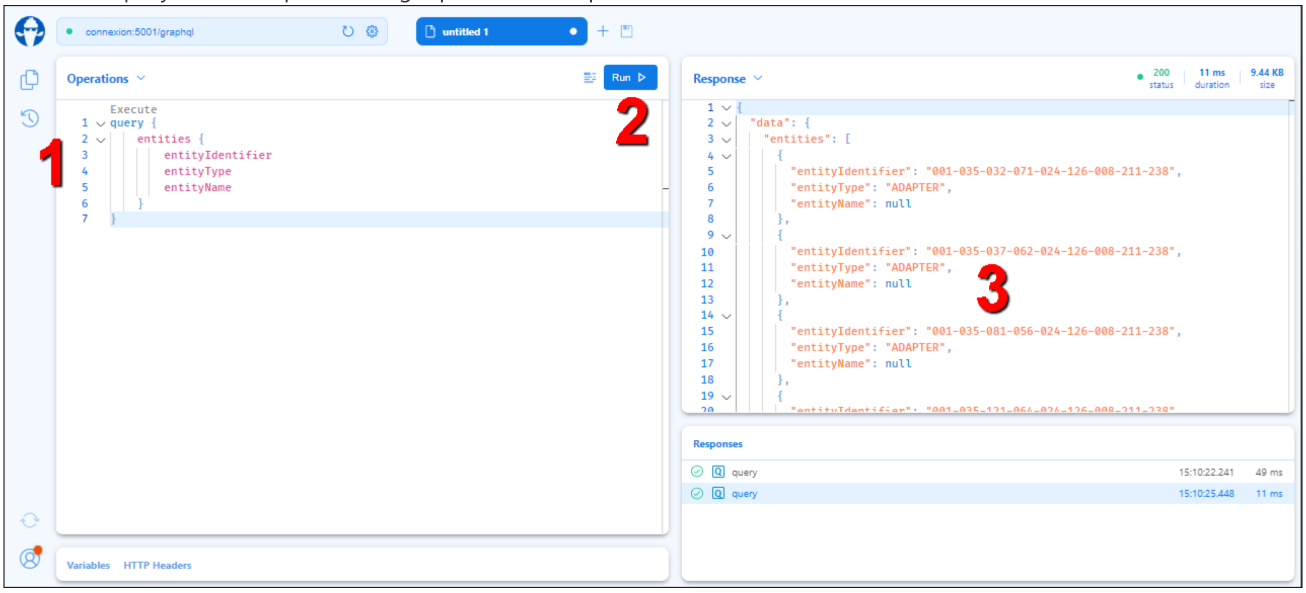
> GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

Source: https://graphql.org/

To get started with GraphQL in ConneX, expand **GraphQL** (in the left navigation pane) and review the available types of data fields you can retrieve from the GraphQL API in ConneX. Then create a list of desired fields and begin to construct queries to retrieve them from ConneX.

To test your queries:

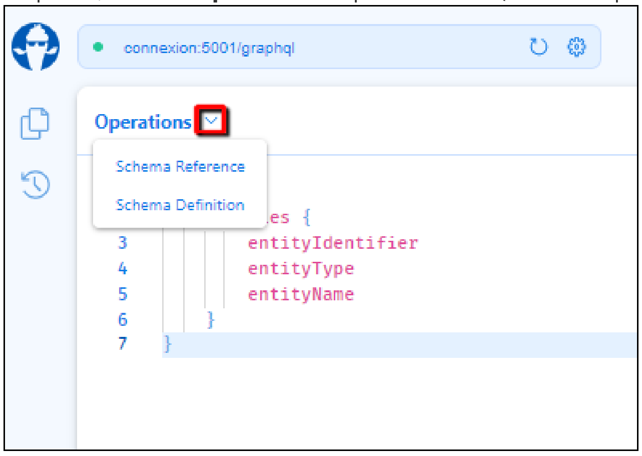
1. Visit http://localhost:5001/graphql from the ConneX Server.
2. Paste your query in the left pane, and then click **Run**.
3. Review the query results/output in the right pane. For example:



```
Execute
query {
  entities {
    entityIdentifier
    entityType
    entityName
  }
}
```

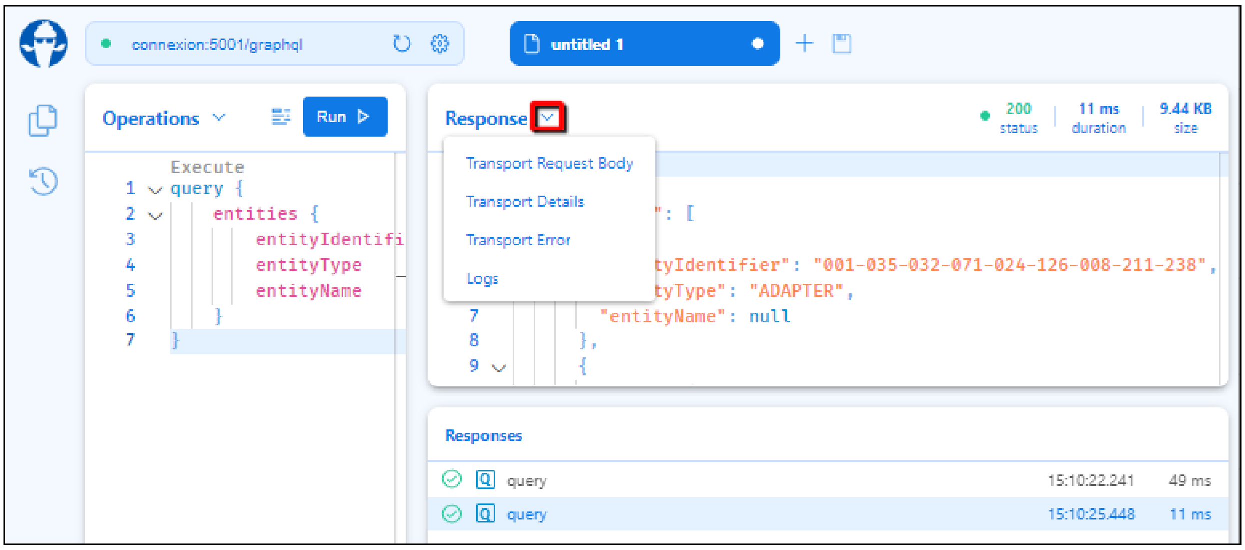
```
{
  "data": {
    "entities": [
      {
        "entityIdentifier": "001-035-032-071-024-126-008-211-238",
        "entityType": "ADAPTER",
        "entityName": null
      },
      {
        "entityIdentifier": "001-035-037-062-024-126-008-211-238",
        "entityType": "ADAPTER",
        "entityName": null
      },
      {
        "entityIdentifier": "001-035-081-056-024-126-008-211-238",
        "entityType": "ADAPTER",
        "entityName": null
      },
      {
        "entityIdentifier": "001-035-121-064-024-126-008-211-238"
```

4. For help on modifying your queries, click the **Operations** drop-down arrow (near the top-left corner) to access the Schema Reference and Definition.



```
        es {
          entityIdentifier
          entityType
          entityName
        }
}
```

5. Similarly, click the **Response** drop-down arrow (in the right pane) for more information about how ConneX processed the query.



```
Execute
query {
  entities {
    entityIdentifi
    entityType
    entityName
  }
}
```

```
": [
tyIdentifier": "001-035-032-071-024-126-008-211-238",
tyType": "ADAPTER",
    "entityName": null
  },
  {
```

```
query    15:10:22.241    49 ms
query    15:10:25.448    11 ms
```

6. After crafting and perfecting your queries as desired, paste them into your manufacturing execution system (MES) or other application and test the queries again from there.

MQTT

ConneX provides an MQTT broker that your MES (or other data processing application) can subscribe to.

What is MQTT?

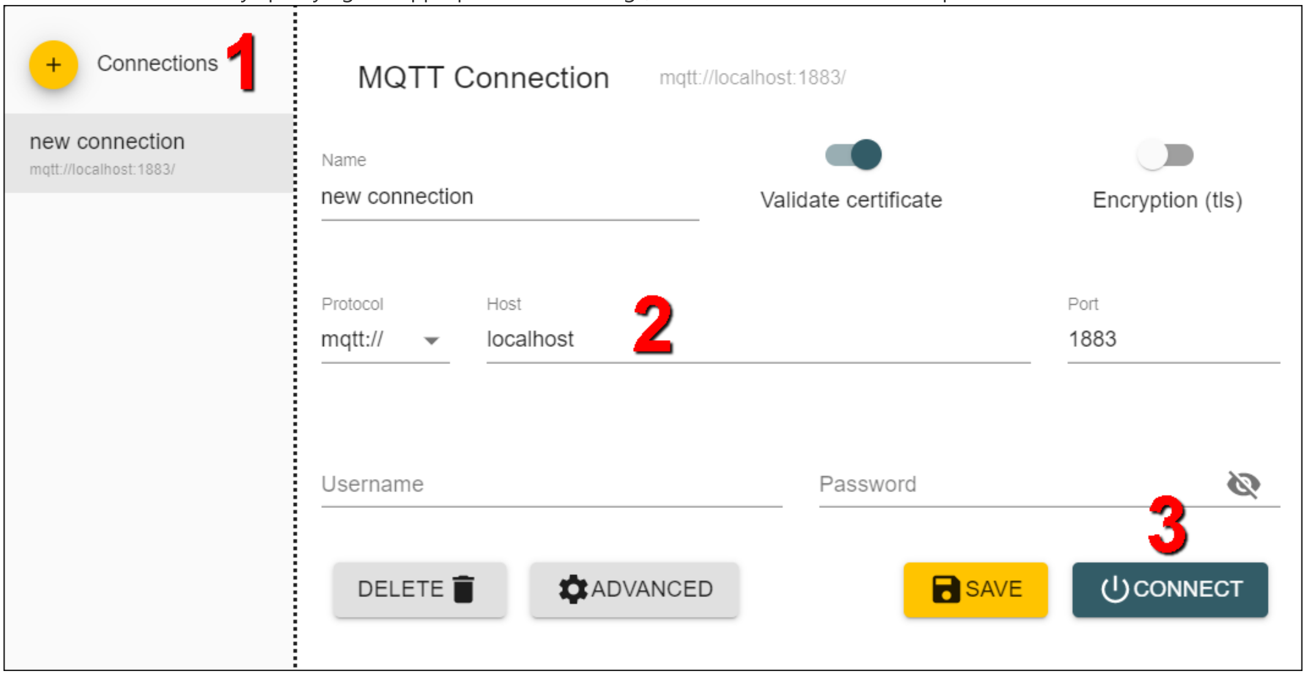
> **MQTT** is a lightweight, publish-subscribe network protocol that transports messages between devices. The protocol usually runs over TCP/IP, however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT. It is designed for connections with remote locations where a resource constraints exist or the network bandwidth is limited. The protocol is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

Source: https://en.wikipedia.org/wiki/MQTT

To get started with MQTT in ConneX, expand **MQTT** (in the left navigation pane) and review the available types of event notifications to which you can subscribe from the MQTT API in ConneX. Then create a list of desired events and begin to construct queries to receive them from ConneX.

To test your queries:

1. Install MQTT Explorer and launch it.
2. Add a new connection by specifying the appropriate host settings, then click **Connect**. For example:

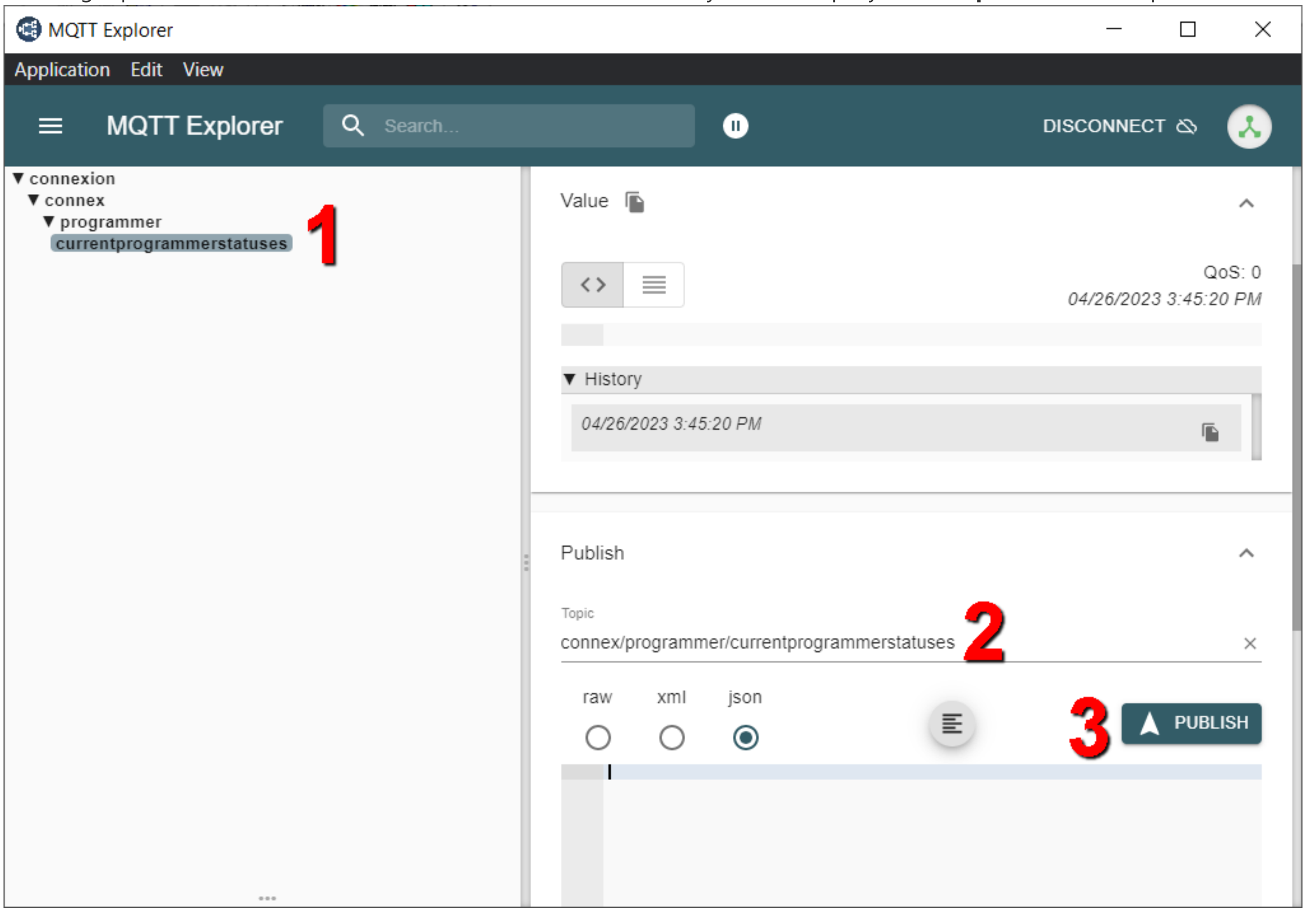


MQTT Connection mqtt://localhost:1883/

new connection
mqtt://localhost:1883/

Name: new connection	Validate certificate: on	Encryption (tls): off
Protocol: mqtt://	Host: localhost	Port: 1883
Username:	Password:	

3. In the left pane treeview, navigate to the desired event.
4. In the right pane, scroll down to the **Publish** section, and confirm/modify the event query in the **Topic** box. For example:



Topic: connex/programmer/currentprogrammerstatuses

raw [✓ json] xml

5. Select a desired output format (raw, xml, or json) and click **Publish** to run the event query.
6. Review the query results/output in the right pane.
7. After crafting and perfecting your queries as desired, paste them into your manufacturing execution system (MES) or other application and test the queries again from there.

Schema

GraphQL

```
schema {
  query: Query
}

type AdapterMetrics {
  id: Long!
  identifier: String
  timeStamp: DateTime!
  programmingDuration: Int!
  verifyDuration: Int!
  blankCheckDuration: Int!
  eraseDuration: Int!
}

"Represents an adapter for a programmer."
type AdapterModel {
  "The database key for the adapter."
  adapterKey: Int!
  "The associated entity for this adapter."
  entity: Entity
  "The last associated programmer for this adapter."
  programmer: ProgrammerModel
  "The adapter's part number identifier."
  adapterId: String
}

type AdapterStatistics {
  adapterId: String
  cleanCount: UnsignedInt!
  lifetimeActuationCount: UnsignedInt!
  lifetimeContinuityFailCount: UnsignedInt!
  lifetimeFailCount: UnsignedInt!
  lifetimePassCount: UnsignedInt!
  socketIndex: Int!
  adapterState: AdapterState!
}

"Information about the offset pagination."
type CollectionSegmentInfo {
  "Indicates whether more items exist following the set defined by the clients arguments."
  hasNextPage: Boolean!
  "Indicates whether more items exist prior the set defined by the clients arguments."
  hasPreviousPage: Boolean!
}

"Represents an abstract component that is connected to the ConneX system."
type Entity {
  "The database key for the entity."
  id: Int!
  "The unique identifier for the entity."
  entityIdentifier: String
  "The type the entity represents."
  entityType: EntityType!
  "The given name of the entity."
  entityName: String
}

"Represents a PSV system connected to ConneX."
type Handler {
  "The database key for the PSV system."
  handlerId: Int!
  "The associated entity for this PSV system."
  entity: Entity
  "The associated programmers for this PSV system."
  programmers: [ProgrammerModel]
  "The PSV system's type (e.g. PSV2800\/3000\/5000\/7000)."
  handlerType: HandlerType!
  "The PSV system's IP address."
  ipAddress: String
  "The PSV system's computer host name."
  hostName: String
  "The PSV system's associated factory."
  machineFactory: String
}

type HandlerMetrics {
  id: Long!
  identifier: String
  timeStamp: DateTime!
  jobState: String
  uptime: Int
  jobProcessingTime: Int
  unitsPerHour: Int
  yield: Float
}

type HandlerStatistics {
  currentJob: String
  availability: Float!
  uptime: String
  totalPass: Int!
  totalFail: Int!
  systemYield: String
  programmerYield: String
  handlerYield: String
  uPH: Int!
  jobCompletionEstimate: String
}

type LicenseModel {
  licenseType: String
  maxConnections: Int!
  availableConnections: Int!
  conneXAnnualMaintenanceContract: DateTime!
  timedLicenseExpiration: DateTime!
}

type MessageModel {
  topic: String
  contentType: String
  timestamp: DateTime!
  messageModelId: UUID!
  payload: [Byte!]
  payloadAsString: String
}

type MessageModelCollectionSegment {
  items: [MessageModel]
  "Information to aid in pagination."
  pageInfo: CollectionSegmentInfo!
  totalCount: Int!
}

type ProgrammerModel {
  programmerId: Int!
  entity: Entity
  handler: Handler
  adapters: [AdapterModel]
  ipAddress: String
  programmerType: ProgrammerType!
}

type Query {
  "Get the last received MQTT message."
  message: MessageModel
  "Get all MQTT messages."
  messages(skip: Int take: Int where: MessageModelFilterInput order: [MessageModelSortInput!]): MessageModelCollectionS
egment
  "Get the latest statistics for the specified adapter."
  latestAdapterStatistics("The adapter's unique identifier." entityIdentifier: String): AdapterStatistics
  "Get the latest statistics for the specified PSV system."
```

```
   Get the latest statistics for the specified PSV system.
  latestHandlerStatistics("The handler system's unique identifier." entityIdentifier: String): HandlerStatistics
  handlerMetrics("The handler's unique identifier." handlerIdentifier: String "The time bucket to aggregate metrics ove
r." timeBucket: String "The interval of time to query." interval: String): [HandlerMetrics]
  adapterMetrics("The adapter's unique identifier." adapterIdentifier: String "The time bucket to aggregate metrics ove
r." timeBucket: String "The interval of time to query." interval: String): [AdapterMetrics]
  "Look up all the known entities connected to this instance of ConneX."
  entities: [Entity]
  "Look up all the known entity types that can be connected to this instance of ConneX."
  entityTypes: [EntityType!]
  "Look up all the known PSV systems connected to this instance of ConneX."
  systems: [Handler]
  "Look up a singular PSV system by its database ID."
  system("The database identifier of the handler." databaseId: Int!): Handler
  "Look up all the known PSV system types that can be connected to this instance of ConneX."
  systemTypes: [HandlerType!]
  "Look up all the known programmers connected to this instance of ConneX."
  programmers: [ProgrammerModel]
  "Look up all the known programmer system types that can be connected to this instance of ConneX."
  programmerTypes: [ProgrammerType!]
  "Look up all the known adapters connected to this instance of ConneX."
  adapters: [AdapterModel]
  "Get the license information for the ConneX Service"
  license: LicenseModel
}

input ComparableByteOperationFilterInput {
  eq: Byte
  neq: Byte
  in: [Byte!]
  nin: [Byte!]
  gt: Byte
  ngt: Byte
  gte: Byte
  ngte: Byte
  lt: Byte
  nlt: Byte
  lte: Byte
  nlte: Byte
}

input ComparableDateTimeOperationFilterInput {
  eq: DateTime
  neq: DateTime
  in: [DateTime!]
  nin: [DateTime!]
  gt: DateTime
  ngt: DateTime
  gte: DateTime
  ngte: DateTime
  lt: DateTime
  nlt: DateTime
  lte: DateTime
  nlte: DateTime
}

input ComparableGuidOperationFilterInput {
  eq: UUID
  neq: UUID
  in: [UUID!]
  nin: [UUID!]
  gt: UUID
  ngt: UUID
  gte: UUID
  ngte: UUID
  lt: UUID
  nlt: UUID
  lte: UUID
  nlte: UUID
}

input ListComparableByteOperationFilterInput {
  all: ComparableByteOperationFilterInput
  none: ComparableByteOperationFilterInput
  some: ComparableByteOperationFilterInput
  any: Boolean
}

input MessageModelFilterInput {
  and: [MessageModelFilterInput!]
  or: [MessageModelFilterInput!]
  topic: StringOperationFilterInput
  contentType: StringOperationFilterInput
  payload: ListComparableByteOperationFilterInput
  timestamp: ComparableDateTimeOperationFilterInput
  messageModelId: ComparableGuidOperationFilterInput
}

input MessageModelSortInput {
  topic: SortEnumType
  contentType: SortEnumType
  timestamp: SortEnumType
  messageModelId: SortEnumType
}

input StringOperationFilterInput {
  and: [StringOperationFilterInput!]
  or: [StringOperationFilterInput!]
  eq: String
  neq: String
  contains: String
  ncontains: String
  in: [String]
  nin: [String]
  startsWith: String
  nstartsWith: String
  endsWith: String
  nendsWith: String
}

enum AdapterState {
  NOT_INSERTED
  INSERTED
  VALIDATED
  VALIDATE_FAILED
  UNKNOWN
  POWER_FAULT
}

enum EntityType {
  HANDLER
  PROGRAMMER
  ADAPTER
  JOB
}

enum HandlerType {
  DESKTOP
  PSV2800
  PSV3000
  PSV5000
  PSV7000
}

enum ProgrammerType {
  FLASH_CORE
  LUMEN_X
}

enum SortEnumType {
  ASC
  DESC
}

"The `@defer` directive may be provided for fragment spreads and inline fragments to inform the executor to delay the e
xecution of the current fragment to indicate deprioritization of the current fragment. A query with `@defer` directive
```

```
will cause the request to potentially return multiple responses, where non-deferred data is delivered in the initial re
sponse and data deferred is delivered in a subsequent response. `@include` and `@skip` take precedence over `@defer`."
directive @defer("If this argument label has a value other than null, it will be passed on to the result of this defer
directive. This label is intended to give client applications a way to identify to which fragment a deferred result bel
ongs to." label: String "Deferred when true." if: Boolean) on FRAGMENT_SPREAD | INLINE_FRAGMENT

"The `@specifiedBy` directive is used within the type system definition language to provide a URL for specifying the be
havior of custom scalar definitions."
directive @specifiedBy("The specifiedBy URL points to a human-readable specification. This field will only read a resul
t for scalar types." url: String!) on SCALAR

"The `@stream` directive may be provided for a field of `List` type so that the backend can leverage technology such as
asynchronous iterators to provide a partial list in the initial response, and additional list items in subsequent respo
nses. `@include` and `@skip` take precedence over `@stream`."
directive @stream("If this argument label has a value other than null, it will be passed on to the result of this strea
m directive. This label is intended to give client applications a way to identify to which fragment a streamed result b
elongs to." label: String "The initial elements that shall be send down to the consumer." initialCount: Int! = 0 "Strea
med when true." if: Boolean) on FIELD

"The `Byte` scalar type represents non-fractional whole numeric values. Byte can represent values between 0 and 255."
scalar Byte

"The `DateTime` scalar represents an ISO-8601 compliant date time type."
scalar DateTime @specifiedBy(url: "https:\/\/www.graphql-scalars.com\/date-time")

"The `Long` scalar type represents non-fractional signed whole 64-bit numeric values. Long can represent values between
-(2^63) and 2^63 - 1."
scalar Long

scalar UUID @specifiedBy(url: "https:\/\/tools.ietf.org\/html\/rfc4122")

"The UnsignedInt scalar type represents a unsigned 32-bit numeric non-fractional value greater than or equal to 0."
scalar UnsignedInt
```

Queries

ConneX exposes the following GraphQL queries:

Query	Description
`adapterMetrics`	Get metrics for the specified adapter.
`adapters`	Look up all the known adapters connected to this instance of ConneX.
`entities`	Look up all the known entities connected to this instance of ConneX.
`entityTypes`	Look up all the known entity types that can be connected to this instance of ConneX.
`handlerMetrics`	Get metrics for the specified PSV system.
`latestAdapterStatistics`	Get the latest metric entries for the specified adapter.
`latestHandlerStatistics`	Get the latest metric entries for the specified PSV system.
`license`	Get the installed license information.
`message`	Get the last received MQTT message.
`messages`	Get all MQTT messages.
`programmers`	Look up all the known programmers connected to this instance of ConneX.
`programmerTypes`	Look up all the known programmer system types that can be connected to this instance of ConneX.
`system`	Look up a singular PSV system by its database ID.
`systems`	Look up all the known PSV systems connected to this instance of ConneX.
`systemTypes`	Look up all the known PSV system types that can be connected to this instance of ConneX.

adapterMetrics

Type: `[AdapterMetrics]`

Get metrics for the specified adapter.

Arguments

Name	Description
`adapterIdentifier` (`String`)	The adapter's unique identifier.
`interval` (`Interval`)	An `Interval` defining how far back to query.
`timeBucket` (`Interval`)	An `Interval` over which the metrics will be aggregated.

Example

Request Response

GraphQL

```
query{
  adapterMetrics(adapterIdentifier: "222-032-205-139-137-224-207-100-238"
    interval: "1 hour"
    timeBucket: "15 minutes")
    {
      timeStamp
      blankCheckDuration
      eraseDuration
      programmingDuration
      verifyDuration
    }
}
```

JSON

```json
{
  "data": {
    "adapterMetrics": [
      {
        "timeStamp": "2022-10-06T11:30:00.000-07:00",
        "blankCheckDuration": 901358,
        "eraseDuration": 896337,
        "programmingDuration": 904021,
        "verifyDuration": 895560
      },
      {
        "timeStamp": "2022-10-06T11:15:00.000-07:00",
        "blankCheckDuration": 895828,
        "eraseDuration": 904213,
        "programmingDuration": 898582,
        "verifyDuration": 898548
      },
      {
        "timeStamp": "2022-10-06T11:00:00.000-07:00",
        "blankCheckDuration": 895486,
        "eraseDuration": 895593,
        "programmingDuration": 902716,
        "verifyDuration": 896741
      },
      {
        "timeStamp": "2022-10-06T10:45:00.000-07:00",
        "blankCheckDuration": 900471,
        "eraseDuration": 897392,
        "programmingDuration": 898490,
        "verifyDuration": 899448
      }
    ]
  }
}
```

adapters

Type: `AdapterModel`

Look up all the known adapters connected to this instance of ConneX.

Example

Request Response

GraphQL

```graphql
query {
  adapters {
    adapterId
  }
}
```

JSON

```json
{
  "data": {
    "adapters": [
      {
        "adapterId": "110008"
      },
      {
        "adapterId": "110008"
      },
      {
        "adapterId": "110008"
      },
      {
        "adapterId": "110008"
      },
      {
        "adapterId": "310008"
      },
      {
        "adapterId": "310008"
      }
    ]
  }
}
```

entities

Type: `[Entity]`

Look up all the known entities connected to this instance of ConneX.

Example

Request Response

GraphQL

```graphql
query {
    entities {
        entityIdentifier
        entityType
        entityName
    }
}
```

JSON

```
{
  "data": {
    "entities": [
      {
        "entityIdentifier": "42707786-1a5b-4b2f-9c0d-9512bb30cbb0",
        "entityType": "HANDLER",
        "entityName": "PSV2800 #1"
      },
      {
        "entityIdentifier": "ee232edf-05ef-4407-a4e4-1d0431099e97",
        "entityType": "HANDLER",
        "entityName": "PSV3000 #1"
      },
      {
        "entityIdentifier": "853be1da-0847-4a85-b499-208c37ce40fb",
        "entityType": "HANDLER",
        "entityName": "PSV5000 #1"
      },
      {
        "entityIdentifier": "777425b0-300e-43d6-b40a-0f94c57559fa",
        "entityType": "HANDLER",
        "entityName": "PSV5000 #2"
      },
      {
        "entityIdentifier": "5573f981-10c4-4466-adf2-c68039cb9983",
        "entityType": "HANDLER",
        "entityName": "PSV7000 #1"
      },
      {
        "entityIdentifier": "ee496e4a-2b14-4cd2-af6a-bb92ad9fa015",
        "entityType": "HANDLER",
        "entityName": "PSV7000 #2"
      },
      {
        "entityIdentifier": "D1:AD:0D:28:26:9E",
        "entityType": "PROGRAMMER",
        "entityName": "FC - 1"
      },
      {
        "entityIdentifier": "D1:AD:0D:28:26:9E_Socket0",
        "entityType": "ADAPTER",
        "entityName": null
      }
    ]
  }
}
```

entityTypes

Type: `[EntityType!]`

Look up all the known entity types that can be connected to this instance of ConneX.

Example

Request Response

GraphQL

```
query {
  entityTypes
}
```

JSON

```
{
  "data": {
    "entityTypes": [
      "HANDLER",
      "PROGRAMMER",
      "ADAPTER",
      "JOB"
    ]
  }
}
```

handlerMetrics

Type: `[HandlerMetrics]`

Get metrics for the specified PSV system.

Arguments

Name	Description
`handlerIdentifier` (`String`)	The PSV system's unique identifier.
`interval` (`Interval`)	An `Interval` defining how far back to query.
`timeBucket` (`Interval`)	An `Interval` over which the metrics will be aggregated.

Example

Request Response

GraphQL

```
query{
  handlerMetrics(handlerIdentifier: "42707786-1a5b-4b2f-9c0d-9512bb30cbb0"
    interval: "1 hour"
    timeBucket: "15 minutes")
    {
      timeStamp
      jobState
      uptime
      jobProcessingTime
      unitsPerHour
      yield
    }
}
```

JSON

```
{
  "data": {
    "handlerMetrics": [
      {
        "timeStamp": "2022-10-06T12:01:57.173-07:00",
        "jobState": null,
        "uptime": 892,
        "jobProcessingTime": null,
        "unitsPerHour": 500,
        "yield": 0.9923533439853912
      },
      {
        "timeStamp": "2022-10-06T11:46:57.173-07:00",
        "jobState": null,
        "uptime": 597,
        "jobProcessingTime": null,
        "unitsPerHour": 503,
        "yield": 0.9923533439853912
      },
      {
        "timeStamp": "2022-10-06T11:31:57.173-07:00",
        "jobState": null,
        "uptime": 1187,
        "jobProcessingTime": 873,
        "unitsPerHour": 498,
        "yield": 0.9923533439853912
      },
      {
        "timeStamp": "2022-10-06T11:16:57.173-07:00",
        "jobState": null,
        "uptime": 890,
        "jobProcessingTime": 884,
        "unitsPerHour": 487,
        "yield": 0.9923533439853912
      }
    ]
  }
}
```

latestAdapterStatistics

Type: `[AdapterStatistics]`

Get the latest statitistics for the specified adapter.

Arguments

Name	Description
`entityIdentifier` (`String`)	The adapter's unique identifier.

Example

Request Response

GraphQL

```
query {
  latestAdapterStatistics(
    entityIdentifier: "136-043-225-168-137-224-207-100-238"
  )
  {
    adapterId
    cleanCount
    lifetimeActuationCount
    lifetimeContinuityFailCount
    lifetimeFailCount
    lifetimePassCount
    socketIndex
    adapterState
  }
}
```

JSON

```
{
  "data": {
    "latestAdapterStatistics": {
      "adapterId": "110008",
      "cleanCount": 2,
      "lifetimeActuationCount": 4955,
      "lifetimeContinuityFailCount": 25,
      "lifetimeFailCount": 70,
      "lifetimePassCount": 9713,
      "socketIndex": 4,
      "adapterState": "VALIDATED"
    }
  }
}
```

latestHandlerStatistics

Type: `[HandlerStatistics]`

Get the latest statistics for the specified PSV system.

Arguments

Name	Description
`entityIdentifier` (`String`)	The PSV system's unique identifier.

Example

Request Response

GraphQL

```
query {
  latestHandlerStatistics(entityIdentifier:"4826196c-0866-44f4-afa0-d331bcfd04eb")
  {
    currentJob
    availability
    uptime
    totalPass
    totalFail
    systemYield
    programmerYield
    handlerYield
    uPH
    jobCompletionEstimate
  }
}
```

JSON

```
{
  "data": {
    "latestHandlerStatistics": {
      "currentJob": "MX 29LV160DBTI",
      "availability": "100.00",
      "uptime": "100.00",
      "totalPass": 189,
      "totalFail": 3,
      "systemYield": "98.44",
      "programmerYield": "98.44",
      "handlerYield": "100.00",
      "uPH": 1859,
      "jobCompletionEstimate": "6/8/2022 11:35:18 AM"
    }
  }
}
```

license

Type: `[LicenseModel]`

Get the installed license information.

Example

Request Response

GraphQL

```
query {
  license {
    licenseType
    maxConnections
    availableConnections
    conneXAnnualMaintenanceContract
    timedLicenseExpiration
  }
}
```

JSON

```
{
  "data": {
    "license": {
      "licenseType": "Perpetual",
      "maxConnections": 10,
      "availableConnections": 9,
      "conneXAnnualMaintenanceContract": "2023-06-01T00:00:00.000-07:00",
      "timedLicenseExpiration": "0001-01-01T00:00:00.000-08:00"
    }
  }
}
```

message

Type: `MessageModel`

Get the last received MQTT message.

Example

Request Response

GraphQL

```
query {
  message
  {
    topic
    timestamp
  }
}
```

JSON

```
{
  "data": {
    "message": {
      "topic": "connex/programmer/currentprogrammerstatuses",
      "timestamp": "2021-12-21T21:48:28.514-08:00"
    }
  }
}
```

messages

Type: `MessageModelCollectionSegment`

Get all MQTT messages using paging.

Arguments

Name	Description
`skip` (`Int`)	The number of messages to skip.
`take` (`Int`)	The number of messages to return.
`where` (MessageModelFilterInput)	The Filter to apply to the messages.
`order` (`[MessageModelSortInput!]`)	The sort order to apply to the messages.

Name	Description

Example

Request Response

GraphQL

```
query {
  messages (take:1) {
    totalCount
    items {
      topic
      contentType
      timestamp
      messageModelId
      payloadAsString
    }
  }
}
```

JSON

```
{
  "data": {
    "messages": {
      "totalCount": 138,
      "items": [
        {
          "topic": "connex/programmer/lumenx/legacy/connected",
          "contentType": null,
          "timestamp": "2022-01-05T21:38:26.835-08:00",
          "messageModelId": "f89a9ea5-cb22-4c32-811e-9fe08f3e6cab",
          "payloadAsString": "{\"ProgrammerIdentifier\":\"\",\"HandlerIdentifer\":\"323bcb6c-3e40-4678-98a8-d373e3
8144af\",\"IpAddress\":\"127.0.0.1\",\"ProgrammerType\":1,\"ProgrammerName\":\"LX-1\",\"Adapters\":[]}"
        }
      ]
    }
  }
}
```

programmers

Type: `[ProgrammerModel]`

Look up all the known programmers connected to this instance of ConneX.

Example

Request Response

GraphQL

```
query {
  programmers {
    programmerId
    ipAddress
    programmerType
  }
}
```

JSON

```
{
  "data": {
    "programmers": [
      {
        "programmerId": 2,
        "ipAddress": "192.168.1.1",
        "programmerType": "FLASH_CORE"
      },
      {
        "programmerId": 3,
        "ipAddress": "192.168.1.2",
        "programmerType": "FLASH_CORE"
      },
      {
        "programmerId": 4,
        "ipAddress": "192.168.1.3",
        "programmerType": "FLASH_CORE"
      },
      {
        "programmerId": 5,
        "ipAddress": "192.168.1.4",
        "programmerType": "FLASH_CORE"
      },
      {
        "programmerId": 6,
        "ipAddress": "192.168.1.5",
        "programmerType": "FLASH_CORE"
      },
      {
        "programmerId": 7,
        "ipAddress": "10.0.0.0",
        "programmerType": "LUMEN_X"
      }
    ]
  }
}
```

programmerTypes

Type: `[ProgrammerType!]`

Look up all the known programmer system types that can be connected to this instance of ConneX.

Example

Request Response

GraphQL

```
query {
  programmerTypes
}
```

JSON

```json
{
  "data": {
    "programmerTypes": [
      "FLASH_CORE",
      "LUMEN_X"
    ]
  }
}
```

system

Type: `Handler`

Look up a singular PSV system by its database ID.

Arguments

Name	Description
`databaseId` (`Int!`)	The database identifier of the handler.

Example

Request   Response

GraphQL

```graphql
query {
    system (databaseId: 1) {
      handlerId
    }
}
```

JSON

```json
{
  "data": {
    "system": {
      "handlerId": 1,
      "handlerType": "PSV2800",
      "ipAddress": "172.16.0.1",
      "hostName": "PSV-1",
      "machineFactory": null
    }
  }
}
```

systems

Type: `[Handler]`

Look up all the known PSV systems connected to this instance of ConneX.

Example

Request   Response

GraphQL

```
query {
    systems {
        handlerId
        handlerType
        ipAddress
        hostName
        machineFactory
    }
}
```

JSON

```
{
  "data": {
    "systems": [
      {
        "handlerId": 1,
        "handlerType": "PSV2800",
        "ipAddress": "172.16.0.1",
        "hostName": "PSV-1",
        "machineFactory": null
      },
      {
        "handlerId": 2,
        "handlerType": "PSV3000",
        "ipAddress": "172.16.0.2",
        "hostName": "PSV-2",
        "machineFactory": null
      },
      {
        "handlerId": 3,
        "handlerType": "PSV5000",
        "ipAddress": "172.16.0.3",
        "hostName": "PSV-3",
        "machineFactory": null
      },
      {
        "handlerId": 4,
        "handlerType": "PSV5000",
        "ipAddress": "172.16.0.11",
        "hostName": "PSV-6",
        "machineFactory": null
      },
      {
        "handlerId": 5,
        "handlerType": "PSV7000",
        "ipAddress": "172.16.0.9",
        "hostName": "PSV-4",
        "machineFactory": null
      },
      {
        "handlerId": 6,
        "handlerType": "PSV7000",
        "ipAddress": "172.16.0.10",
        "hostName": "PSV-5",
        "machineFactory": null
      }
    ]
  }
}
```

systemTypes

Type: `[HandlerType!]`

Look up all the known PSV system types that can be connected to this instance of ConneX.

Example

Request Response

GraphQL

```graphql
query {
  systemTypes
}
```

JSON

```json
{
  "data": {
    "systemTypes": [
      "DESKTOP",
      "PSV2800",
      "PSV3000",
      "PSV5000",
      "PSV7000"
    ]
  }
}
```

Objects

ConneX exposes the following GraphQL objects:

AdapterMetrics

Represents metrics related to an adapter.

Fields

Name	Description
`blankCheckDuration` (`Int!`)	The duration for a blank check operation (in milliseconds).
`eraseDuration` (`Int!`)	The duration for an erase operation (in milliseconds).
`id` (`Long!`)	The metric's index.
`identifier` (`String!`)	The entity identifier for the metric.
`programmingDuration` (`Int!`)	The duration for a programming operation (in milliseconds).
`timeStamp` (`DateTime!`)	The time stamp for the metric.
`verifyDuration` (`Int!`)	The duration for a verify operation (in milliseconds).

AdapterModel

Represents an adapter for a programmer.

Fields

Name	Description
`adapterKey` (`Int!`)	The database key for the adapter.
`entity` (`Entity`)	The associated entity for this adapter.
`programmer` (`ProgrammerModel`)	The last associated programmer for this adapter.
`adapterId` (`String`)	The adapter's part number identifier.

AdapterStatistics

Represents an adapter's statistics.

Fields

Name	Description
`adapterId` (`String`)	The adapter's ID.
`cleanCount` (`UnsignedInt!`)	The adapter's clean count.
`lifetimeActuationCount` (`UnsignedInt!`)	The adapter's lifetime actuation count.
`lifetimeContinuityFailCount` : (`UnsignedInt!`)	The adapter's lifetime continuity fail count.
`lifetimeFailCount` (`UnsignedInt!`)	The adapter's lifetime fail count.
`lifetimePassCount` (`UnsignedInt!`)	The adapter's lifetime pass count.
`socketIndex` (`Int!`)	The adapter's socket index.
`adapterState` (`AdapterState!`)	The adapter's `AdapterState`.

CollectionSegmentInfo

Information about the offset pagination.

Fields

Name	Description
`hasNextPage` (`Boolean!`)	Indicates whether more items exist following the set defined by the clients arguments.

Name	Description
`hasPreviousPage` (`Boolean!`)	Indicates whether more items exist prior the set defined by the clients arguments.

Entity

Represents an abstract component that is connected to the ConneX system.

Fields

Name	Description
`id` (`Int!`)	The database key for the entity.
`entityIdentifier` (`String`)	The unique identifier for the entity.
`entityType` (`EntityType!`)	The type the entity represents.
`entityName` (`String`)	The given name of the entity.

Handler

Represents a PSV system connected to ConneX.

Fields

Name	Description
`handlerId` (`Int!`)	The database key for the PSV system.
`entity` (`Entity`)	The associated entity for this PSV system.
`programmers` (`[ProgrammerModel]`)	The associated programmers for this PSV system.
`handlerType` (`HandlerType!`)	The PSV system's type (e.g. PSV2800/3000/5000/7000).
`ipAddress` (`String`)	The PSV system's IP address.
`hostName` (`String`)	The PSV system's computer host name.
`machineFactory` (`String`)	The PSV system's associated factory.

HandlerMetrics

Represents metrics related to a handler system.

Fields

Name	Description
`id` (`Long!`)	The metric's index.
`identifier` (`String`)	The entity identifier for the metric.
`jobProcessingTime` (`String`)	The job processing time since the last measurement.
`jobState` (`String`)	The job's current state.
`timeStamp` (`DateTime!`)	The time stamp for the metric.
`unitsPerHour` (`Int`)	The system's UPH (units per hour).
`uptime` (`Int`)	The uptime of the system since the last measurement.
`yield` (`Float`)	The system's programmer yield.

HandlerStatistics

Represents a handler system's statistics.

Fields

Name	Description
`currentJob` (`String`)	The current job the system is running.
`availability` (`Float!`)	The system's availability.
`uptime` (`String`)	The system's uptime.
`totalPass` (`Int!`)	The total number of passed devices.

Name	Description
`totalFail` (`Int!`)	The total number of failed devices.
`systemYield` (`String`)	The system's yield.
`programmerYield` (`String`)	The system's programmer yield.
`handlerYield` (`String`)	The system's handler yield.
`uPH` (`Int!`)	The system's UPH (units per hour).
`jobCompletionEstimate` (`String`)	The system's job completion estimate.

Interval

Represents a time interval as a `String`. Interval values can be written using the following syntax:

`quantity unit`

where `quantity` is an `Int!` and `unit` is one of the following:

Unit	ISO 8601 Abbreviation	Example
`microsecond`		`1 microsecond(s)`
`millisecond`		`1 millisecond(s)`
`second`	`S`	`1 second(s)`
`minute`	`M` (in the time part)	`1 minute(s)`
`hour`	`H`	`1 hour(s)`
`day`	`D`	`1 day(s)`
`week`	`W`	`1 week(s)`
`month`	`M` (in the date part)	`1 month(s)`
`year`	`Y`	`1 year(s)`
`decade`		`1 decade(s)`
`century`		`1 century(ies)`
`millenium`		`1 millenium(s)`

LicenseModel

Represents the license information for the ConneX Service.

Fields

Name	Description
`availableConnections` (`Int!`)	The number of connections still available for use.
`conneXAnnualMaintenanceContract` (`DateTime!`)	The expiration date for the ConneX Annual Maintenence Contract.
`licenseType` (`String`)	The type of license installed.
`maxConnections` (`Int!`)	The maximum number of connections available with the installed license.
`timedLicenseExpiration` (`DateTime!`)	The expiration date of the license (if applicable).

The field `licenseType` has the following values:

Value	Description
`NoLicense`	No ConneX license has been installed.
`Perpetual`	License is perpetual for the purchased version.
`Timed`	License is time bound based on purchase agreement.

MessageModel

Represents a message received over MQTT.

Fields

Name	Description
`topic` (`String`)	The MQTT topic.
`contentType` (`String`)	The MQTT message type.
`timestamp` (`DateTime!`)	The message timestamp.
`messageModelId` (`UUID!`)	The unique UUID message identifier.

Name	Description
`payload` (`[Byte!]`)	The message payload in raw bytes.
`payloadAsString` (`String`)	The message payload converted to a `UTF8` string.

MessageModelCollectionSegment

Represents a collection of MessageModel (used in pagination).

Fields

Name	Description
`items` (`[MessageModel]`)	The items in the current page.
`pageInfo` (`CollectionSegmentInfo!`)	Information to aid in pagination.
`totalCount` (`Int!`)	The total message count for the query.

ProgrammerModel

Represents a programmer connected to ConneX.

Fields

Name	Description
`programmerId` (`Int!`)	The database key for the programmer.
`entity` (`Entity`)	The associated entity for this programmer.
`handler` (`Handler`)	The associated handler for this programmer.
`adapters` (`[AdapterModel]`)	A collection of adapters associated with this programmer.
`ipAddress` (`String`)	The IP address of the programmer.
`programmerType` (`ProgrammerType!`)	The type of programmer.

Interfaces

ConneX exposes the following GraphQL interfaces:

None

Enums

ConneX exposes the following GraphQL enums:

AdapterState

Represents the different state an adapter can be in.

Values	Description
NOT_INSERTED	Adapter is not inserted.
INSERTED	Adapter is inserted.
VALIDATED	Adapter is inserted and validated.
VALIDATE_FAILED	Adapter validation failed.
UNKNOWN	Adapter state is unknown.
POWER_FAULT	Adapter experienced a power fault.

EntityType

Represents the different types an entity can represent.

Values	Description
HANDLER	Represents a PSV system.
PROGRAMMER	Represents a programmer (e.g. LumenX or FlashCORE).
ADAPTER	Represents a programmer adapter.
JOB	Represents a programming job.

HandlerType

Represents the different types a PSV system object can be.

Values	Description
DESKTOP	Represents a desktop programming system.
PSV2800	Represents a PSV2800 programming system.
PSV3000	Represents a PSV3000 programming system.
PSV5000	Represents a PSV5000 programming system.
PSV7000	Represents a PSV7000 programming system.

ProgrammerType

Represents the different types a programmer object can be.

Values	Description
FLASH_CORE	Represents a FlashCORE III programmer.
LUMEN_X	Represents a LumenX programmer.

SortEnumType

Represents the different types of sorting that can be applied when filtering.

Values	Description
ASC	Sort the values in ascending order.
DESC	Sort the values in descending order.

Objects

ConneX exposes the following GraphQL input objects:

ComparableByteOperationFilterInput

Represents filters for a `Byte` type.

Input Fields

Name	Description
`eq` (`Byte`)	Filter results to when the `Byte` value equals the given value.
`gt` (`Byte`)	Filter results to when the `Byte` value is greater than the given value.
`gte` (`Byte`)	Filter results to when the `Byte` value is greater than or equal to the given value.
`in` (`[Byte!]`)	Filter results to when the `Byte` value is in the collection of the given values.
`lt` (`Byte`)	Filter results to when the `Byte` value is less than the given value.
`lte` (`Byte`)	Filter results to when the `Byte` value is less than or equal to the given value.
`neq` (`Byte`)	Filter results to when the `Byte` value does *not* equals the given value.
`ngt` (`Byte`)	Filter results to when the `Byte` value is *not* greater than the given value.
`ngte` (`Byte`)	Filter results to when the `Byte` value is *not* greater than or equal to the given value.
`nin` (`[Byte!]`)	Filter results to when the `Byte` value is *not* in the collection of the given values.
`nlt` (`Byte`)	Filter results to when the `Byte` value is *not* less than the given value.
`nlte` (`Byte`)	Filter results to when the `Byte` value is *not* less than or equal to the given value.

ComparableDateTimeOperationFilterInput

Represents filters for a `DateTime` type.

Input Fields

Name	Description
`eq` (`DateTime`)	Filter results to when the `DateTime` value equals the given value.
`gt` (`DateTime`)	Filter results to when the `DateTime` value is greater than the given value.
`gte` (`DateTime`)	Filter results to when the `DateTime` value is greater than or equal to the given value.
`in` (`[DateTime!]`)	Filter results to when the `DateTime` value is in the collection of the given values.
`lt` (`DateTime`)	Filter results to when the `DateTime` value is less than the given value.
`lte` (`DateTime`)	Filter results to when the `DateTime` value is less than or equal to the given value.
`neq` (`DateTime`)	Filter results to when the `DateTime` value does *not* equals the given value.
`ngt` (`DateTime`)	Filter results to when the `DateTime` value is *not* greater than the given value.
`ngte` (`DateTime`)	Filter results to when the `DateTime` value is *not* greater than or equal to the given value.
`nin` (`[DateTime!]`)	Filter results to when the `DateTime` value is *not* in the collection of the given values.
`nlt` (`DateTime`)	Filter results to when the `DateTime` value is *not* less than the given value.
`nlte` (`DateTime`)	Filter results to when the `DateTime` value is *not* less than or equal to the given value.

ComparableGuidOperationFilterInput

Represents filters for a `UUID` type.

Input Fields

Name	Description
`eq` (`UUID`)	Filter results to when the `UUID` value equals the given value.
`gt` (`UUID`)	Filter results to when the `UUID` value is greater than the given value.
`gte` (`UUID`)	Filter results to when the `UUID` value is greater than or equal to the given value.
`in` (`[UUID!]`)	Filter results to when the `UUID` value is in the collection of the given values.
`lt` (`UUID`)	Filter results to when the `UUID` value is less than the given value.
`lte` (`UUID`)	Filter results to when the `UUID` value is less than or equal to the given value.
`neq` (`UUID`)	Filter results to when the `UUID` value does *not* equals the given value.

Name	Description
`ngt` (`UUID`)	Filter results to when the `UUID` value is *not* greater than the given value.
`ngte` (`UUID`)	Filter results to when the `UUID` value is *not* greater than or equal to the given value.
`nin` (`[UUID!]`)	Filter results to when the `UUID` value is *not* in the collection of the given values.
`nlt` (`UUID`)	Filter results to when the `UUID` value is *not* less than the given value.
`nlte` (`UUID`)	Filter results to when the `UUID` value is *not* less than or equal to the given value.

ListComparableByteOperationFilterInput

Represents filters for a `[Byte]` type.

Input Fields

Name	Description
`all` (`ComparableGuidOperationFilterInput`)	Filter results to when all match the given. `ComparableGuidOperationFilterInput`
`any` (`Boolean`)	TODO: Figure out what this does.
`none` (`ComparableGuidOperationFilterInput`)	Filter results to when none match the given `ComparableGuidOperationFilterInput`.
`some` (`ComparableGuidOperationFilterInput`)	Filter results to when some match the given `ComparableGuidOperationFilterInput`.

MessageModelFilterInput

Represents filters for a `[MessageModel]` type.

Input Fields

Name	Description
`and` (`[MessageModelFilterInput]`)	Add additional filtering criteria to restrict results.
`contentType` (`StringOperationFilterInput`)	Filter results based on the `contentType` field.
`messageModelId` (`ComparableGuidOperationFilterInput`)	Filter results based on the `messageModelId`
`or` `[MessageModelFilterInput]`	Add additional filtering criteria to expand results.
`payload` (`ListComparableByteOperationFilterInput`)	Filter results based on the `payload` field.
`timestamp` (`ComparableDateTimeOperationFilterInput`)	Filter results based on the `timestamp` field.
`topic` (`StringOperationFilterInput`)	Filter results based on the `topic` field.

MessageModelSortInput

Represents sort orders for a `[MessageModel]` type.

Input Fields

Name	Description
`contentType` (`SortEnumType`)	Sort results based on the `contentType` field.
`messageModelId` (`SortEnumType`)	Sort results based on the `messageModelId`
`timestamp` (`SortEnumType`)	Sort results based on the `timestamp` field.
`topic` (`SortEnumType`)	Sort results based on the `topic` field.

StringOperationFilterInput

Represents filters for a `String` type.

Input Fields

Name	Description
`and` (`[StringOperationFilterInput!]`)	Add additional filtering criteria to restrict results.
`contains` (`String`)	Filter results to when the `String` value contains the given value.
`endsWith` (`String`)	Filter results to when the `String` value ends with the given value.

Name	Description
`eq` (`String`)	Filter results to when the `String` value equals the given value.
`in` (`[String]`)	Filter results to when the `Byte` value is in the collection of the given values.
`ncontains` (`String`)	Filter results to when the `String` value does *not* contain the given value.
`nendsWith` (`String`)	Filter results to when the `String` value does *not* end with the given value.
`neq` (`String`)	Filter results to when the `String` value does *not* equals the given value.
`nin` (`[String]`)	Filter results to when the `String` value is *not* in the collection of the given values.
`nstartswith` (`[String]`)	Filter results to when the `String` value does *not* start with the given value.
`or` (`[StringOperationFilterInput!]`)	Add additional filtering criteria to expand results.
`startswith` (`[String]`)	Filter results to when the `String` value starts with the given value.

Scalars

ConneX exposes the following GraphQL scalars:

Name	Description
Boolean	The `Boolean` scalar type represents `true` or `false`.
Byte	The `Byte` scalar type represents non-fractional whole numeric values. Byte can represent values between 0 and 255.
DateTime	The `DateTime` scalar represents an ISO-8601 compliant date time type.
Float	The `Float` scalar type represents signed double-precision fractional values as specified by IEEE 754.
Int	The `Int` scalar type represents non-fractional signed whole numeric values. Int can represent values between -(2^31) and 2^31 - 1.
Long	The `Long` scalar type represents non-fractional signed whole 64-bit numeric values. Long can represent values between -(2^63) and 2^63 - 1.
String	The `String` scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.
UUID	A field whose value is a generic Universally Unique Identifier.

Automated Handling software

Below, you will find the MQTT events that are published/subscribed relating to:

- AH700
- CH700

AH700

AH700 software is used to control the following Data I/O handling system:

- PSV7000

CH700

CH700 software is used to control the following Data I/O handling systems:

- PSV5000
- PSV3500
- PSV3000

Events

References to "x" (in "xh700" and "xhsessionid") below, should be replaced with "a" or "c" when subscribing to AH700 or CH700 topics and replaced with "A" or "C" when retrieving the version from the AH700 version field (xH700Version).

Below are the events that PSV systems publish:

Event	Description
Begin Job Session	Event fired when xH700 begins running a job.
Device Complete	Event fired after the handler places a device in the output media.
Device Inspection	Event fired after the handler inspects a part at 2D and/or 3D station.
End Job Session	Event fired when xH700 finishes running a job.
Light Tower Status	Event fired when the light tower state changes.
Marking	Event fired after the handler marks a part.
Pick Part	Event fired when the handling system picks up a part.
Place Part	Event fired when the handler places a part.
Shutdown	Event fired when xH700 shuts down gracefully, after xH700 is past its splash screen.
Startup	Event fired after xH700 "Start" button is pressed.
System Statistics	Event fired periodically providing the current system statistics.
System Status	Event fired when the status changes in xH700.
User Created	Event fired when an xH700 user is created.
User Deleted	Event fired when an xH700 user is deleted.
User Login	Event fired when a user attempts to log in to xH700.
User Logout	Event fired when a user logs out of xH700.

Begin Job Session

Topic: `xh700/beginrun/{hostname}/{xhsessionid}`

Event fired when xH700 begins running a job.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`2DInspectionProjectFile` (`string`)	The full path of the file used for 2D inspection.
`3DInspectionProjectFile` (`string`)	The full path of the file used for 3D inspection.
`IgnoreProgrammers` (`string`)	`True` if programmers are ignored, otherwise `False`.
`LaserMarkingProjectFile` (`string`)	The full path of the file used for laser marking

Name	Description
`MachineID` (`string`)	The machine identifier.
`MachineParametersFile` (`string`)	The contents of the file used for machine parameters.
`PackageParametersFile` (`string`)	The contents of the file used for package parameters.
`Sumcheck` (`string`)	The checksum for the job that is starting.
`TaskName` (`string`)	The name of the job that is starting.
`VisionInspectionProjectFile` (`string`)	(CH700 only) The full path of the file used for vision inspection.
`WinAH400INIFile` (`string`)	The contents of the `WinAH400.ini` file used.
`xH700Version` (`string`)	The version of the installed xH700 software.

Device Complete

Topic: `xh700/devicecomplete/{hostname}/{xhsessionid}`

Event fired after the handler places a device in the output (pass or fail) media.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DeviceID` (`ulong`)	The identification number assigned by the handling system.
`HandlerErrorCode` (`enum`)	The `HandlerErrorCode` of the handler.
`Status` (`string`)	The `OperationStatus` indicating the state of the device.

Device Inspection

Topic: `xh700/operations/inspection/{hostname}/{xhsessionid}`

Event fired after the handler inspects a part at 2D and/or 3D station.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DeviceID` (`ulong`)	The unique device identifer.
`InspectionResult` (`string`)	A `InspectionResult` representing the result of the device inspection.
`PickHead` (`uint`)	The pick head end effector.

End Job Session

Topic: `xh700/endrun/{hostname}/{xhsessionid}`

Event fired when xH700 finishes running a job.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DevicesFailedOn3DSystem` (`int`)	The number of devices that failed as a result of the 3D system.
`DevicesFailedOnLaser` (`int`)	The number of devices that failed as a result of the laser.
`DevicesFailedOnProgrammers` (`int`)	The number of devices that failed as a result of programming.
`DevicesFailedREST` (`uint`)	The number of devices that failed recurrent empty socket test (REST).
`DevicesFailedVision` (`uint`)	The number of devices that failed vision inspection.

Name	Description
`DevicesMissingInUse` (`uint`)	The number of devices missing in use.
`DevicesPickedInput` (`ulong`)	The number of devices picked from the input media.
`EndingSerialNumber` (`string`)	The ending serial number for the job session.
`FailQuantity` (`ulong`)	The number of devices that failed in the job session.
`IgnoreProgrammers` (`string`)	`True` if programmers are ignored, otherwise `False` .
`InputMedia` (`string`)	The input location media type.
`JobAssistanceTime` (`string`)	The job assistance time.
`JobProcessingTime` (`string`)	The job processing time.
`JobThroughput` (`ulong`)	The job throughput of the job session.
`NominalThroughput` (`double`)	The nominal throughput of the job session.
`OutputMedia` (`string`)	The output location media type.
`PassQuantity` (`ulong`)	The number of devices that passed in the job session.
`Reject1` (`string`)	The Reject1 location media type.
`Reject2` (`string`)	The Reject2 location media type.
`SerialFailReport` (`string`)	The number of devices using serialization that failed.
`SerialPassReport` (`string`)	The number of devices using serialization that passed.
`StartingSerialNumber` (`string`)	The starting serial number for the job session.
`TerminationReason` (`string`)	Provides a reason as to why the job session ended.

Light Tower Status

Topic: `xh700/lighttowerchanged/{hostname}/{xhsessionid}`

Event fired when the light tower state changes.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`NewState` (`string`)	The new light tower `TowerState` indicated below
`OldState` (`string`)	The old light tower `TowerState` indicated below

Marking

Topic: `xh700/operations/marking/{hostname}/{xhsessionid}`

Event fired after the handler marks a part.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Cup` (`uint`)	The cup the device was marked on.
`DeviceID` (`ulong`)	The unique device identifer.
`Status` (`string`)	The `OperationStatus` indicating the result of a marking operation.

Pick Part

Topic: `xh700/operations/pick/{hostname}/{xhsessionid}`

Event fired when the handler picks up a part.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DeviceID` (`ulong`)	The unique device identifer.
`Location` (`string`)	The `Location` the device was picked from.
`PickHead` (`uint`)	The pick head end effector used to pick the device.
`Position` (`uint`)	The position within the given location.
`Status` (`string`)	The `OperationStatus` indicating the result of the pick operation.

Place Part

Topic: `xh700/operations/place/{hostname}/{xhsessionid}`

Event fired when the handler places a part.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DeviceID` (`ulong`)	The unique device identifer.
`Location` (`string`)	The `Location` the device was picked from.
`PickHead` (`uint`)	The pick head end effector used to pick the device.
`Position` (`uint`)	The position within the given location.
`Status` (`string`)	The `OperationStatus` indicating the result of the place operation.

Shutdown

Topic: `xh700/shutdown/{hostname}/{xhsessionid}`

Event fired when xH700 shuts down gracefully, after xH700 is past its splash screen.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Active` (`bool`)	Always false.

Special Properties

- Last Will and Testament

Startup

Topic: `xh700/startup/{hostname}/{xhsessionid}`

Event fired after xH700 "Start" button is pressed.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Active` (`bool`)	Always false.
`MachineType` (`enum`)	The `MachineType` representing the type of machine that is running.

Special Properties

- Retained

System Statistics

Topic: `xh700/systemstatistics/{hostname}/{xhsessionid}`

Event fired periodically providing the current system statistics.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`DevicesFailedOn3DSystem` (`int`)	The number of devices that failed as a result of the 3D system.
`DevicesFailedOnLaser` (`int`)	The number of devices that failed as a result of the laser.
`DevicesFailedOnProgrammer` (`int`)	The number of devices that failed as a result of programming.
`DevicesFailedREST` (`int`)	The number of devices that failed recurrent empty socket test (REST).
`DevicesFailedVision` (`int`)	The number of devices that failed vision inspection.
`DevicesPickedInput` (`int`)	The number of devices picked from the input media.
`HandlerYield` (`string`)	The percentage of devices that were picked from the input media and are placed in the output and reject media.
`JobAssistanceTime` (`string`)	The job assistance time.
`JobCompletionEstimate` (`string`)	The estimated job completion time.
`JobProcessingTime` (`string`)	The job processing time.
`ProgrammerYield` (`string`)	The percentage of devices that passed programming.
`SystemYield` (`string`)	The percentage of devices that were picked from the input media and are placed in the output media.
`TotalFail` (`int`)	The number of devices that failed in the job session.
`TotalPass` (`int`)	The number of devices that passed in the job session.
`UPH` (`int`)	The job throughput (including operator intervention time) of the job session.

System Status

Topic: `xh700/systemstatus/{hostname}/{xhsessionid}`

Event fired when the status of xH700 changes.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`ErrorMessage` (`ErrorMessage`)	The `ErrorMessage`) of the handler.
`RunState` (`enum`)	The `RunState` of the handler.

User Created

Topic: `xh700/users/create/{hostname}/{xhsessionid}`

Event fired when an xH700 user is created.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Role` (`enum`)	The user's `UserRole`
`Username` (`string`)	Username of the newly created user.

User Deleted

Topic: `xh700/users/delete/{hostname}/{xhsessionid}`

Event fired when an xH700 user is deleted.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Username` (`string`)	Username of the deleted user.

User Login

Topic: `xh700/users/login/{hostname}/{xhsessionid}`

Event fired when a user attempts to log in to xH700.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`UserSessionID`	The session ID for the current user of xH700.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Success` (`bool`)	Indicates whether or not the login was successful.
`Username` (`string`)	Username of the user who attempted to log in.

User Logout

Topic: `xh700/users/logout/{hostname}/{xhsessionid}`

Event fired when a user logs out of xH700.

Level	Description
`hostname`	The hostname of the PC that xH700 is running on.
`xhsessionid`	The session ID for the current instance of xH700.

Fields

Name	Description
`Username` (`string`)	Username of the user who logged out.

Commands

References to "x" (in "xh700" and "xhsessionid") below, should be replaced with "a" or "c" when sending commands to AH700 or CH700.

Below are the commands that can be used to control xH700:

Command	Description
Abort Job Session	Instruct xH700 to end a job session.
Pause Job Session	Instruct xH700 to pause a job session.

Abort Job Session

Topic: `command/xh700/abortjob/{hostname}/{xhsessionid}`

Instruct xH700 to end a job session. xH700 does not pick anymore devices from input, finishes the current devices in the work-envelope, then stops.

Level	Description
`hostname`	The hostname of the PC that should be paused.
`xhsessionid`	The session ID that should be paused.

Pause Job Session

Topic: `command/xh700/pausejob/{hostname}/{xhsessionid}`

Instruct xH700 to pause a job session.

Level	Description
`hostname`	The hostname of the PC that should be paused.
`xhsessionid`	The session ID that should be paused.

Types

Types that different fields can return.

HandlerErrorCode

Type: `enum`

Represents the possible handler error codes assigned to rejected devices.

State	Description
`3`	Device programming error.
`12`	Recurrent Empty Socket Test (REST) error; The device is not programmed.
`15`	Device laser marking error.
`17`	Device continuity error.
`200`	Device position inspection error.
`201`	Device 3D inspection error.

InspectionResult

Type: `string`

Represents the possible return values of a device inspection operation:

Fail Code	Description
`201`	3D inspection failure.
`901`	2D inspection failure, unexpected device in pocket.
`902`	2D inspection failure, device not detected.
`903`	2D inspection failure, device detected but failed inspection.
`904`	2D inspection timeout failure.
`PASS 2D`	Device passed 2D inspection
`PASS 3D`	Device passed 3D inspection

Location

Type: `string`

Represents a physical location inside a PSV system.

Location	Description
`Laser`	The laser marking device.
`Programmer`	A programmer (e.g. FlashCORE or LumenX).
`Tape`	An input/output tape.
`Tray`	An input/output tray.
`Tube`	An input/output tube.

MachineType

Type: `enum`

Represents the type of PSV system.

Location	Description
`Desktop Mode`	System is running in desktop mode.
`PSV3000`	System is a PSV3000 machine.
`PSV5000`	System is a PSV5000 machine.
`PSV7000`	System is a PSV7000 machine.

OperationStatus

Type: `string`

Represents the possible return values for a pass/fail operation

Position	Description
`Fail`	Operation was unsuccessful.
`Pass`	Operation was successful.

TowerState

Type: `string`

Represents the possible return values for tower statuses.

State	Description
`Alternating Green-Yellow`	Light tower is alternating between green and yellow lamps illuminated.
`Alternating Yellow-Red`	Light tower is alternating between yellow and red lamps illuminated.
`Alternating Yellow-Red with alarm`	Light tower is alternating the yellow and red lamp and emitting an audible alarm.
`Flashing Red`	Light tower is flashing the red lamp exclusively.
`Flashing red with alarm`	Light tower is flashing the red lamp exclusively and emitting an audible alarm.
`Flashing Yellow`	Light tower is flashing the yellow lamp exclusively.
`Flashing Yellow with alarm`	Light tower is flashing the yellow lamp exclusively and emitting an audible alarm.
`Green`	Light tower has only the green lamp illuminated.
`Off`	Light tower is off.
`Red`	Light tower has only the Red lamp illuminated.
`Yellow`	Light tower has only the yellow lamp illuminated.

UserRole

Type: `enum`

Represents the possible return values for a user's role.

Position	Description
`Operator`	User has operator level permissions.
`Service`	User has service level permissions.
`Supervisor`	User has supervisor level permissions.

RunState

Type: `enum`

Represents the possible return values for the job run state of xH700.

State	Description
`JobIdle`	The job has not been started.
`JobPaused`	The job is paused.
`JobRunning`	The job is running.
`JobStopped`	The handler encountered an error and cannot continue the job.

DMS

Below, you will find the MQTT events that are published/subscribed relating to DMS:

Message	Description
Begin Download	Event fired when DMS begins a job download.
End Download	Event fired when DMS completes a job download.

Begin Download

Topic: `dms/jobs/begindownload/{hostname}/{jobname}`

Event fired when DMS begins a job download.

Level	Description
hostname	The hostname of the PC that DMS is running on.
jobname	The name of the job.

Fields

Name	Description
DownloadSize (`ulong`)	The size of the job in bytes.

End Download

Topic: `dms/jobs/enddownload/{hostname}/{jobname}`

Event fired when DMS begins a job download.

Level	Description
hostname	The hostname of the PC that DMS is running on.
jobname	The name of the job.

Fields

Name	Description
Success (`bool`)	Whether or not the download was sucessful.

Programmer

Below, you will find the MQTT events that are published/subscribed relating to a programmer:

Message	Description
Adapter Inserted	Event fired when an adapter is inserted into a programmer.
Adapter Removed	Event fired when an adapter is removed from a programmer.
Programmer Connected	Event fired when a programmer connects to the system.
Programmer Offline	Event fired when a programmer goes offline.
Programmer Online	Event fired when a programmer comes online.
Programmer Removed	Event fired when a programmer disconnects from the system.
Programming Complete	Event fired when a programmer completes a programming cycle.

Adapter Inserted

Topic: `programmers/adapter/inserted/{programmerserialnumber}/{adapterserialnumber}`

Event fired when an adapter is inserted into a programmer.

Level	Description
programmerserialnumber	The programmer's unique serial number.
adapterserialnumber	The adapter's unique serial number.

Fields

Name	Description
AdapterId (`string`)	The adapter's identifier (e.g. 110008).
AdapterIndex (`int`)	The adapter's index position.
SocketInformation (`[SocketInfo]`)	The socket information for the adapter.

Adapter Removed

Topic: `programmers/adapter/removed/{programmerserialnumber}/{adapterserialnumber}`

Event fired when an adapter is inserted into a programmer.

Level	Description
programmerserialnumber	The programmer's unique serial number.
adapterserialnumber	The adapter's unique serial number.

Fields

Name	Description
AdapterId (`string`)	The adapter's identifier (e.g. 110008).
AdapterIndex (`int`)	The adapter's index position.

Programmer Connected

Topic: `connex/programmer/{programmertype}/legacy/connected`

Event fired when a programmer establishes a connection to the system.

Level	Description
programmertype	The type of programmer that connected to the system: **LumenX** or **FlashCore**.

Fields

Name	Description
Adapters (`[AdapterInformation]`)	The adapters that are inserted in the programmer.
HandlerIdentifier (`string`)	The Unique ID of the system handler that the programmer connected to.
IpAddress (`string`)	The programmer's IP address.

Name	Description
ProgrammerIdentifier (`string`)	The Unique ID of the specific programmer that connected to the system.
ProgrammerName (`string`)	The name of the programmer that connected to the system.
ProgrammerType (`string`)	The type of programmer that connected to the system: **LumenX** or **FlashCore**.

Programmer Offline

Topic: `programmers/poweroff/{programmerserialnumber}`

Event fired when a programmer is powered off.

Level	Description
programmerserialnumber	The programmer's unique serial number.

Programmer Online

Topic: `programmers/poweron/{programmerserialnumber}`

Event fired when a programmer is powered on.

Level	Description
programmerserialnumber	The programmer's unique serial number.

Fields

Name	Description
Adapters (`[AdapterInformation]`])	The adapters that are insert in the programmer.
AdditionalInformation (`Dictionary<string, string>`)	The additional information for the programmer.
IpAddress (`string`)	The programmer's IP address.
ProgrammerType (`string`)	The programmer's type.
VersionInformation (`[ProgrammerVersionInformation]`])	The version information for the different programmer components.

Programmer Removed

Topic: `connex/programmer/{programmertype}/legacy/removed`

Event fired when a programmer disconnects from the system.

Level	Description
programmertype	The type of programmer that disconnected from the system: **LumenX** or **FlashCore**.

Fields

Name	Description
HandlerIdentifier (`string`)	The Unique ID of the system handler that the programmer disconnected from.
IpAddress (`string`)	The programmer's IP address.
ProgrammerIdentifier (`string`)	The Unique ID of the specific programmer that disconnected from the system.
ProgrammerName (`string`)	The name of the programmer that disconnected from the system.
ProgrammerType (`string`)	The type of programmer that disconnected to the system: **LumenX** or **FlashCore**.

Programming Complete

Topic: `connex/programmer/{programmertype}/legacy/programmingcomplete`

Event fired when a programmer completes a data and/or security provisioning cycle of the part/device, thereby producing a device record.

Level	Description
programmertype	The type of programmer performing the data and/or security provisioning: **LumenX** or **FlashCore**.

Fields (at minimum)

Name	Description
TimeStamp	Date and Time (in UTC) of the programming event.
ProgrammerClass	Type of programming unit (LumenX or FlashCore).
ProgrammerFirmwareVersion	Firmware version of the programming unit.
ProgrammerSerialNumber	Serial number of the programming unit.
ProgrammerSystemVersion	System version of the programming unit.
ProgrammerIP	IP address of the programming unit.
AdapterId	Unique ID of the socket adapter on the programming unit.
AdapterSerialNumber	Serial Number of the socket adapter on the programming unit.
AdapterCleanCount	Number of times the "clean adapter module" reminder message was displayed to Operators.
AdapterLifetimeActuationCount	Total number of times the socket adapter is mechanically actuated over the life of the adapter.
AdapterLifetimeContinuityCount	Total number of devices that ran continuity check over the life of the adapter.
AdapterLifetimeContinuityFailCount	Total number of devices that failed continuity check over the life of the adapter.
AdapterLifetimeFailCount	Total number of devices that failed to complete all operations of a job over the life of the adapter.
AdapterPassCount	Total number of devices that passed all operations of a job over the life of the adapter.
AdapterSocketIndex	Index number of a particular socket adapter.
AdapterState	The state of the socket adapter.
AlgorithmID	Unique ID that specifies the particular algorithm used in the job.
JobID	Unique ID that specifies the particular job.
JobName	Name of the job.
JobDescription	Description of the job.
DeviceName	Name of the device.
DeviceManufacturer	Name of the device manufacturer.
ChipID	Unique Chip ID on the device.
RawChipID	Raw Chip ID on the device.
SocketIndex	Index number of the socket adapter in which the device was placed and programmed.
Code	Unique status code that represents Pass, Fail, or Other.
CodeName	Name of the result code/status.
ProgramDuration	Amount of time elapsed to complete programming.
VerifyDuration	Amount of time elapsed to verify programming.
TimesTime	Total time minus the time for Blank Check, Erase, Program, and Verify operations.
AlgoDeviceDetailsCID	Unique Chip ID correlating the algorithm with the device.
BlankCheckDuration	Amount of time elapsed to perform the Blank Check operation.
EraseDuration	Amount of time elapsed to perform the Erase operation.
ErrorMessage	Specific message describing the error.
SerialData	Serialization pattern to be programmed into devices.

Types

Types that different fields can return.

AdapterInformation

Represents information about a programming adapter.

Fields

Name	Description
SerialNumber (`string`)	The adapter's unique serial number.
AdapterId (`string`)	The adapter's identifier (e.g. 110008).
AdapterIndex (`int`)	The adapter's index position.
SocketInformation (`[SocketInfo]`])	The socket information for the adapter.

ProgrammerVersionInformation

Represents versioning information about a programmer component.

Fields

Name	Description

Name	Description
VersionName (`string`)	The name of the component name.
Version (`string`)	The programmer's version.

SocketInfo

Represents statistical information about a programming socket.

Fields

Name	Description
CleanCount (`string`)	The adapter's clean count.
LifetimeActuationCount (`uint`)	The adapter's lifetime actuation count.
LifetimeContinuityFailCount (`uint`)	The adapter's lifetime continuity fail count.
LifetimeFailCount (`uint`)	The adapter's lifetime fail count.
LifetimePassCount (`uint`)	The adapter's lifetime pass count.

Machine Manager

Below you will find the events and commands that are published/subscribed relating to the Machine Manager service.

Commands

Below are the commands that can be sent to a Machine Mangager instance:

Command	Description
`Launch DMS`	Launches DMS for use with LumenX programming.
`Launch TaskLink`	Launches TaskLink for use with FlashCORE programming.

Launch DMS

Topic: `command/dms/launchdms/{hostname}`

Instruct the Machine Manager service to launch DMS for use with LumenX programming.

Level	Description
`hostname`	The hostname of the PC that should launch DMS.

A `CommandResponse` will be published on the topic `machinemananger/commandresponse/{hostname]` indicating the success (or failure) of the command.

Fields

Name	Description
`JobName` (`string`)	Set the selected job by name to run.
`JobPath` (`string`)	Set the selected job by file path to run.
`Quantity` (`int`)	Must be a whole, non-zero number. Sets the number of devices to process when this Job runs.

Note
Providing both `JobName` and `JobPath` fields is not supported and will result in an error.

Launch TaskLink

Topic: `command/tasklink/launchtasklink/{hostname}`

Instruct the Machine Manager service to launch TaskLink for use with FlashCORE programming.

Level	Description
`hostname`	The hostname of the PC that should launch TaskLink.

Fields

Name	Description
`TaskName` (`string`)	Run the specified Task and exit TaskLink. The Task must be present in the current Task file.
`AdministratorMode` (`bool`)	Run TaskLink in Administrator Mode.
`BatchMode` (`bool`)	Run TaskLink in Batch Mode.
`Quantity` (`int`)	Must be a whole, non-zero number. Sets the number of devices to process when this Job runs. This option disables the prompt for pass quantity at run-time and is useful for Batch Mode operation.

Note
The `TaskName` field can also be used to launch with specific database such as `task_database::task_name`. See the TaskLink documentation for more information.

A `CommandResponse` will be published on the topic `machinemananger/commandresponse/{hostname]` indicating the success (or failure) of the command.

Common Types

The following types are shared across the different software components:

Types

Type	Description
`CommandResponse`	Represents a result of a command message.
`ErrorMessage`	Represents an error message.

CommandResponse

Represents a result of a command message

Fields

Name	Description
CommandTopic `string`	The command topic that was sent.
ErrorMessage (`ErrorMessage`)	If not successful, the accompanying error message.
Success (`bool`)	Indicates if the command was successful or not.

ErrorMessage

Represents an error message.

Fields

Name	Description
ErrorCode (`string`)	The error code (if provided).
ErrorLevel (`ErrorLevel`)	The error level.
Message (`string`)	The error message.

Enumerations

Enumeration	Description
`ErrorLevel`	The severity level of error.

ErrorLevel

The severity level of error.

Value	Description
0 - Warning	The error is a warning, but operation can continue.
1 - Error	The error is an error, operation cannot continue.
2 - Fatal	The error is a fatal error, operation cannot continue and software may be in an unstable state.