ENISCO FORCAM
smart factory experts

FORCE EDGE CONNECT

FORCE EDGE CONNECT

Version: 230721

Manual

Document:	Manual - FORCE EDGE CONNECT
Release date:	2023-07-21
Document version:	1
Author:	FORCAM GmbH

Contents

1 About this document

This document describes how to use FORCE EDGE CONNECT (hereafter simply referred to as EDGE CONNECT).
The manual describes the different components and functions that are available for connecting assets and the possibilities the application offers for signal interpretation and transmission.

Target group

The manual requires, among other, basic knowledge in signal processing and electronic data processing as well as certain knowledge regarding the controllers and communication protocols used.
If you do not have any knowledge in this area, take the time to familiarize yourself with the basics.

ⓘ We recommend that you use our Academy: **https://forcam.com/academie/**
The FORCAM Academy provides the knowledge to effectively use the methods for digital transformation and the technologies for the Smart Factory.
Based on lean manufacturing and TPM methods, our institute team will guide you to initiate changes in the company and to use the technologies correctly.

ⓣ In our **Customer Area** you can find all manuals and product descriptions as well as additional information on your release. Additional information on graphical signal composition can be found in the **Manual – Graphical Composition** there.

2 Concept

FORCE EDGE CONNECT (hereafter simply referred to as EDGE CONNECT) offers manufacturing companies a solution for digitizing almost any asset, regardless of its age or technical state. An asset is a generic term for all objects that EDGE CONNECT can connect, such as machines, sensors, data beacons and IT systems. Thus, FORCAM supports the digital transformation of manufacturing processes in the Green- and Brownfield environment.

FORCAM therefore delivers a product that addresses the main requirement of Industry 4.0 by extracting digital information from the production asset. This closes the gap between IT (information technology) and OT (operative technology).

EDGE CONNECT offers a variety of possible methods to connect assets, it sends the asset signals to superordinate systems by means of standardized events. These can be ME (Manufacturing Execution) or MOM (Manufacturing Operation Management) systems such as SAP DM/ME or MII, among others. FORCAM can thus reduce the time and effort required for digitization and create a standardized interface to the machine park. The assets are connected using plug-ins. Many common manufacturer-specific (proprietary) protocols are presently supported (such as HEIDENHAIN, Siemens S7 or FANUC & Co.) as well as many common communication standards (such as MTConnect, OPC UA or MQTT). In case an asset is not network-capable, the FORCAM I/O Controller is available as separate hardware for digitizing such assets.

The asset connections are used to obtain a wide variety of information. This includes information about the current status asset or their sensor readings such as temperatures, pressures or energy consumption. In the Brownfield as well as in the Greenfield environment it is important not only to read the signals and pass them on, but also to interpret them for further processing. This task is performed by the Signal Composition component. This makes it possible, for example, to find out when an asset is actually in production or at a standstill. Another essential part of the solution is the handling of NC programs and the possibility to transfer them to and from asset.

The modern and also cleanly structured menu navigation of EDGE CONNECT makes it possible to digitally connect asset in a quick and efficient way using the available control and signal information.

The Machine Repository (optional extension) makes it easy to create and use templates. Templates are used for recurring settings when connecting assets. They already contain all the important general information. Only individual information, such as IP address or serial number, must be added. With the MR, you can either define templates for asset connections or derive a template from an existing connection and use it to further connect assets of the same type. The template structure ensures a standardized connection of identical assets, thus enabling the comparison of assets of the same type. This further reduces the individual effort required to connect an asset, enabling the time- and resource-efficient implementation of digitization projects.

EDGE CONNECT is flexible and can be applied to any manufacturing company. The individual components of the solution can be located in different areas and levels and provide benefits at each level.

3 System components

This chapter describes the individual EDGE CONNECT components and their functions.

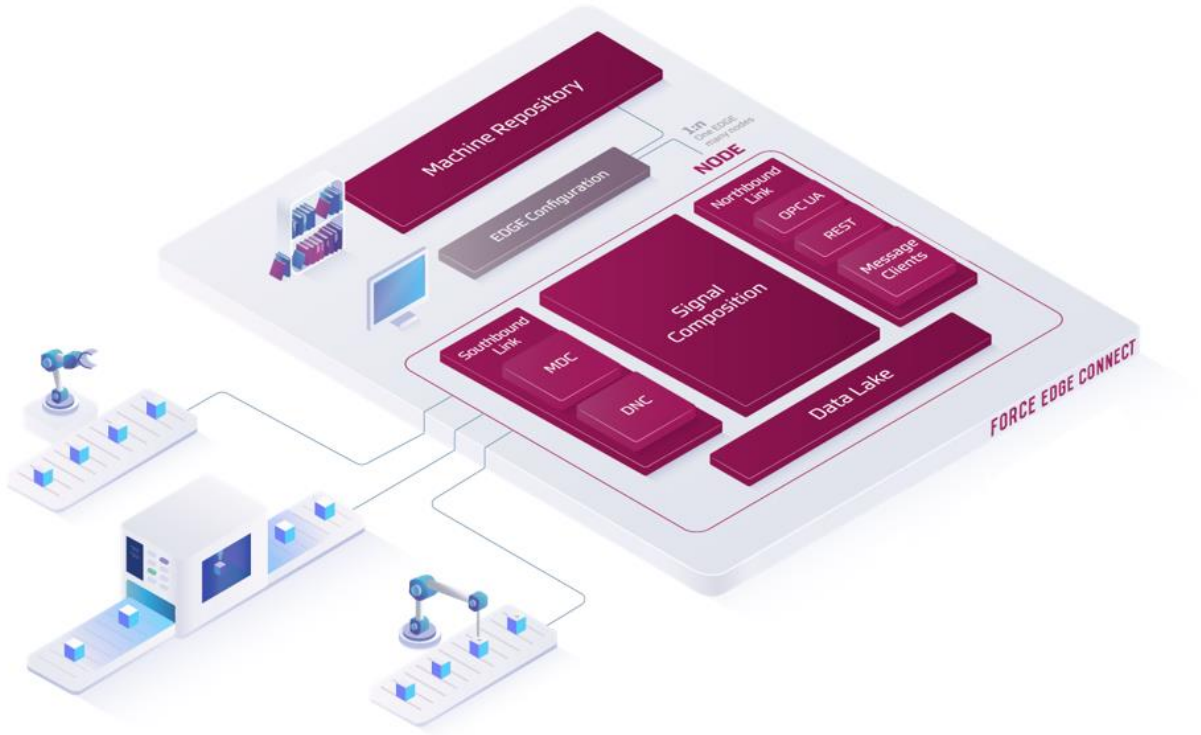


Fig. 1: Schematic structure of EDGE CONNECT

3.1 EDGE Node

The EDGE NODE is the central element of EDGE CONNECT when it comes to connecting assets. It consists of the following key components:

3.1.1 Southbound Link

The Southbound Link component is responsible for the communication between EDGE CONNECT and the asset. In terms of infrastructure, EDGE CONNECT is located above the asset level (shopfloor). This is why we refer to the communication between assets and EDGE CONNECT as “southbound” communication.

Plug-ins

The plug-ins used in the EDGE CONNECT establish communication links with specific machine controllers. They also standardize the data, thus making evaluations more comparable.
They allow for direct communication with various asset controllers, but also cover modern communication protocols such as MQTT, UPC UA and many more.
The plug-ins are divided into those for Machine Data Collection (MDC) and for Distributed Numerical Control (DNC).

- MDC plug-ins for machine data collection

  These include plug-ins designed for unidirectional reading of machine signals as well as plug-ins for bidirectional signal transmission, i.e., for reading and writing back signals.
- DNC plug-ins for transferring and reading NC files
  These plug-ins can be used to transfer NC programs to the machine's file system or to query the program that is active on the machine.

⚠ EDGE CONNECT is not intended to be used for providing, editing, or managing NC programs.

For the most common control types, a set of plug-ins is included in EDGE CONNECT by default. The annex contains a list of all FORCAM plug-ins that are currently available.

3.1.2 Signal Composition

This component is used to derive logical asset states. This allows standardized events to be derived from signal combinations. Events are messages that are sent to a third-party system.
Signal composition also makes it possible to react to events and to write values to the control unit of the asset (if this is supported by control unit and protocol).
Such a composition can be implemented in EDGE CONNECT either via a script or using a graphical solution. The graphical composition provides an easy introduction into the world of signal composition. (For more information about this editor, see the **Manual - Graphical Composition**).

3.1.3 Northbound Link

The Northbound Link makes asset data from the Signal Composition in EDGE CONNECT available to any third-party system. In terms of infrastructure, the 3rd party system is located above EDGE CONNECT. This is why we refer to the communication between EDGE CONNECT and 3rd party systems as “northbound” communication.

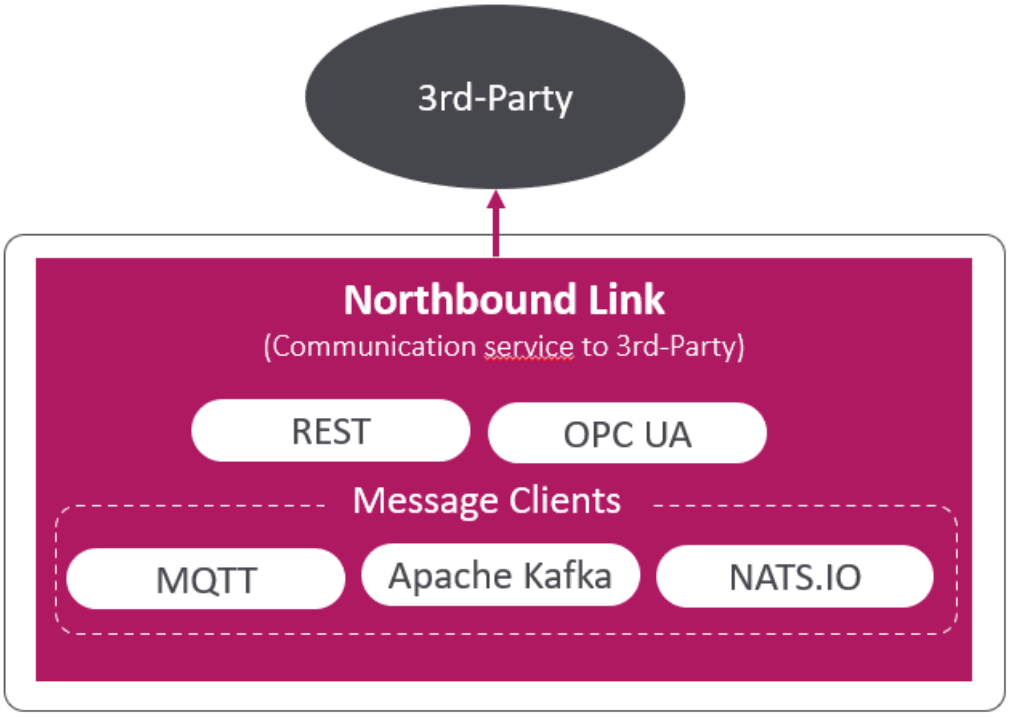


Fig. 2: Northbound Link

The Northbound Link component is used to forward asset data to superordinate systems (3rd party systems) in the form of standardized events. The following options are available for connecting superordinate systems:

- HTTP/REST
- MQTT
- Apache Kafka
- OPC UA
- NATS.io

The message content can be configured for each connection and event. If MQTT, NATS.io or Apache Kafka are used, a broker is required as middleware.
The Northbound Link is delivered with preconfigured standard events for communication with the MES or ERP level. If necessary, these can be further individualized.

⚠ The middleware (broker) must be provided and configured separately. It is not part of the EDGE CONNECT.

Data Lake

To obtain a digital twin of an asset or control unit, it is not only important to establish the connection to the asset, interpret the signals and pass them on to other applications, but also to store the data. With the Data Lake component, all data is stored at signal level, at signal interpretation level (Signal Composition) and at event level. This includes changes in the configuration, write operations and transferred NC files. Data is made available via the Data Lake API. This allows the latest AI algorithms, visualization tools, but also audit requirements to benefit. The Data Lake is designed as short-term data memory.

ⓘ The Data Lake component must be purchased separately, in addition to EDGE CONNECT.

3.2 EDGE Configuration

EDGE Configuration is the user interface for EDGE CONNECT. It can be used to manage multiple EDGE nodes. An EDGE node is the bundling of signal collection from several assets. Depending on the amount of data, one or more EDGE nodes are used per plant. Node administration is done centrally in the EDGE Configuration.

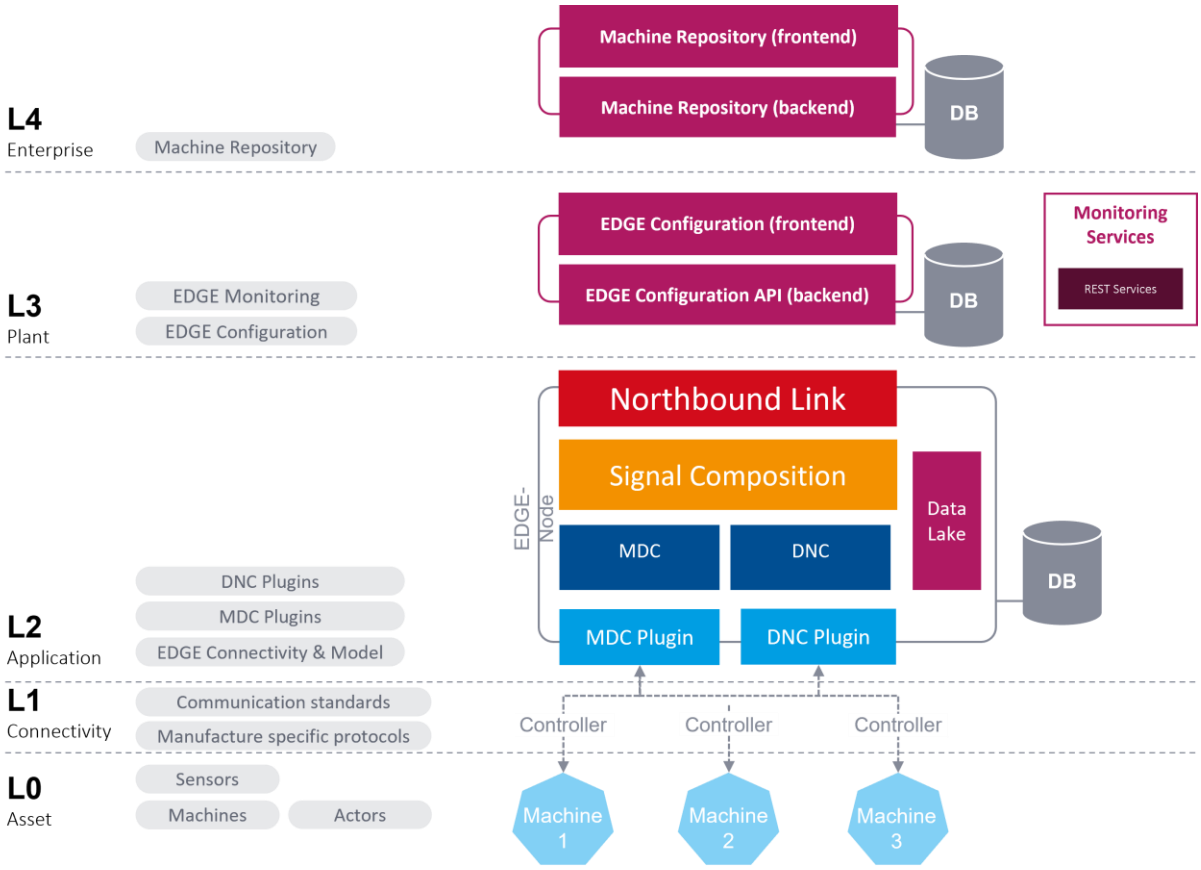
3.3 Machine Repository

The Machine Repository allows templates to be generated from existing asset connections or for new ones. These templates can be used to connect assets of the same type and the same usage type in a standardized manner. The template contains all configuration elements that are not asset-specific. Asset and connection-specific configuration elements are, for example, IP address, serial number, equipment number, etc. In addition, templates lead to a standardized and unified asset configuration, which makes data more comparable when it comes to evaluation.

⚠ The Machine Repository must be purchased separately in addition to EDGE CONNECT.

3.4 System architecture

EDGE CONNECT is architecturally divided into levels (layers). These are based on the business use case, which enables a high scalability of the individual components. For example, multiple EDGE nodes can be hosted to divide the assets logically, but also based on performance.



Level 0 - Assets

The lowest layer is where the assets, sensors and actuators are connected.

Level 1 - Connectivity

The growing selection of plug-ins facilitates the connection of a wide variety of controllers with their different communication standards such as OPC UA or MT Connect, as well as manufacturer-specific protocols.

Level 2 - Application

The number of possible EDGE nodes is not limited. A node encompasses several layers or tasks.

Level 3 - Plant

The configuration component can serve 1 (minimum) to n EDGE nodes. This can be provided per plant, as well as per production line.
Each component can run independently and without an active connection to other components of EDGE CONNECT (e.g., due to temporary loss). This enables a wide variety of deployment. For example, EDGE Configuration does not necessarily have to be hosted in EDGE CONNECT itself, but only a connection to the respective API must to be established.

All components communicate via standardized interfaces (HTTP/REST).

Level 4 - Enterprise

The Machine Repository is an EDGE CONNECT extension that lets you create and manage asset connection templates.

4 Deployment

Fig. 3: Options for installing EDGE CONNECT

For the installation, an installer is provided in which the EDGE Configuration, the EDGE Node and (optionally) the Machine Repository are included. These are installed either by the customer or by a FORCAM service provider.
EDGE Configuration contains the entire user interface including all functions.
EDGE Node contains the EDGE node and can be installed as often as needed, since the number of nodes is only limited by the license. The maximum number of nodes depends on the chosen subscription model. This defines how many nodes can be created and how many assets can be connected per node.
The Machine Repository contains the user interface along with its functions. It can support a large number of EDGE instances. An EDGE instance is composed of an EDGE Configuration with its associated EDGE Nodes.

⚠ The Machine Repository component must be purchased separately, in addition to EDGE CONNECT.

ⓘ For more information, see the **Installation Guides** and the **System Requirements** documents.

5 Basic settings

General settings for EDGE CONNECT are made in the menu (Fig. 4).
Besides introducing the handling of tables, this chapter provides information on the following aspects:

- User administration
- Supplied master data
- Licensing
- Profile
- Download area
- Monitoring
- Logging out

ⓘ The settings made for language and dark mode in the profile are saved in the user profile and apply only to this user.

Fig. 4:Calling the menus in EDGE CONNECT

5.1 User administration

Users for EDGE CONNECT are created in the user administration. Each user can be assigned permissions containing only the functions appropriate or intended for that user (e.g. configure asset, restart node, etc.). Existing user accounts can also be edited subsequently.

ⓘ Once the permissions of a logged-in user have been changed, they take effect immediately after a new login. However, it may take up to 30 minutes for the change to take effect if the user does not log in again.

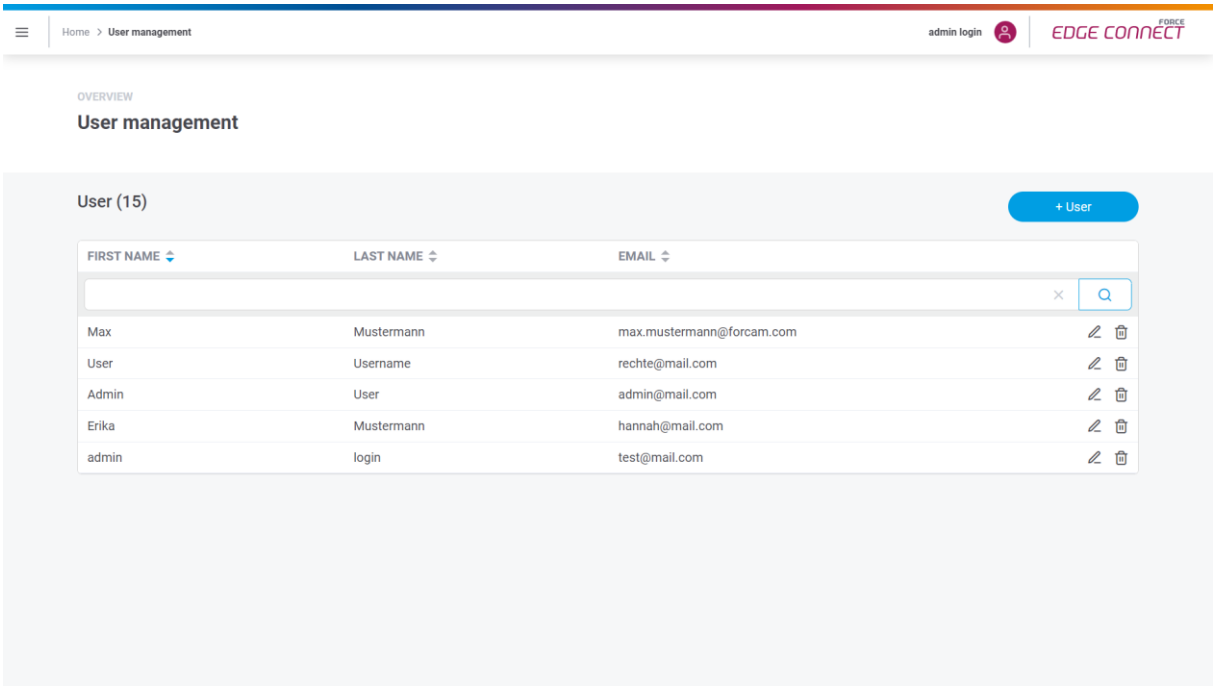


Fig. 5: EDGE CONNECT user administration with 5 users

To create a new user:

1. Click on **+ User**.
2. Enter an email address, first and last name in the subsequent dialog.
3. Set the desired password.
   This must be at least 8 characters long, consist of upper - and lower-case letters and contain at least one number and one special character.
   The following special characters are permitted:
   ! „ # $ % & , ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
4. Assign user permissions (see Table 1).
5. Save.

ⓘ i A user with the same data cannot be created a second time.

User permission	Description
User administration	The user can call-up the user administration, create new users and assign/remove permissions.
Monitoring page	The user can view the monitoring area from the EDGE configuration. Changes cannot be made.
Restart Edge node	The user can restart an EDGE node.
Supplied master data	The user can transfer and connect asset master data from a third-party system to EDGE CONNECT.
Configure nodes	The user can edit or delete EDGE Nodes and make changes to the event configurations.
Configure without template	The user can create assets without the use of MR templates.
Change template-based values	The user can modify MR templates (signals and script).
Configure with template	The user can create assets only by means of MR templates. Changes cannot be made.

Table 1: User permissions in EDGE CONNECT

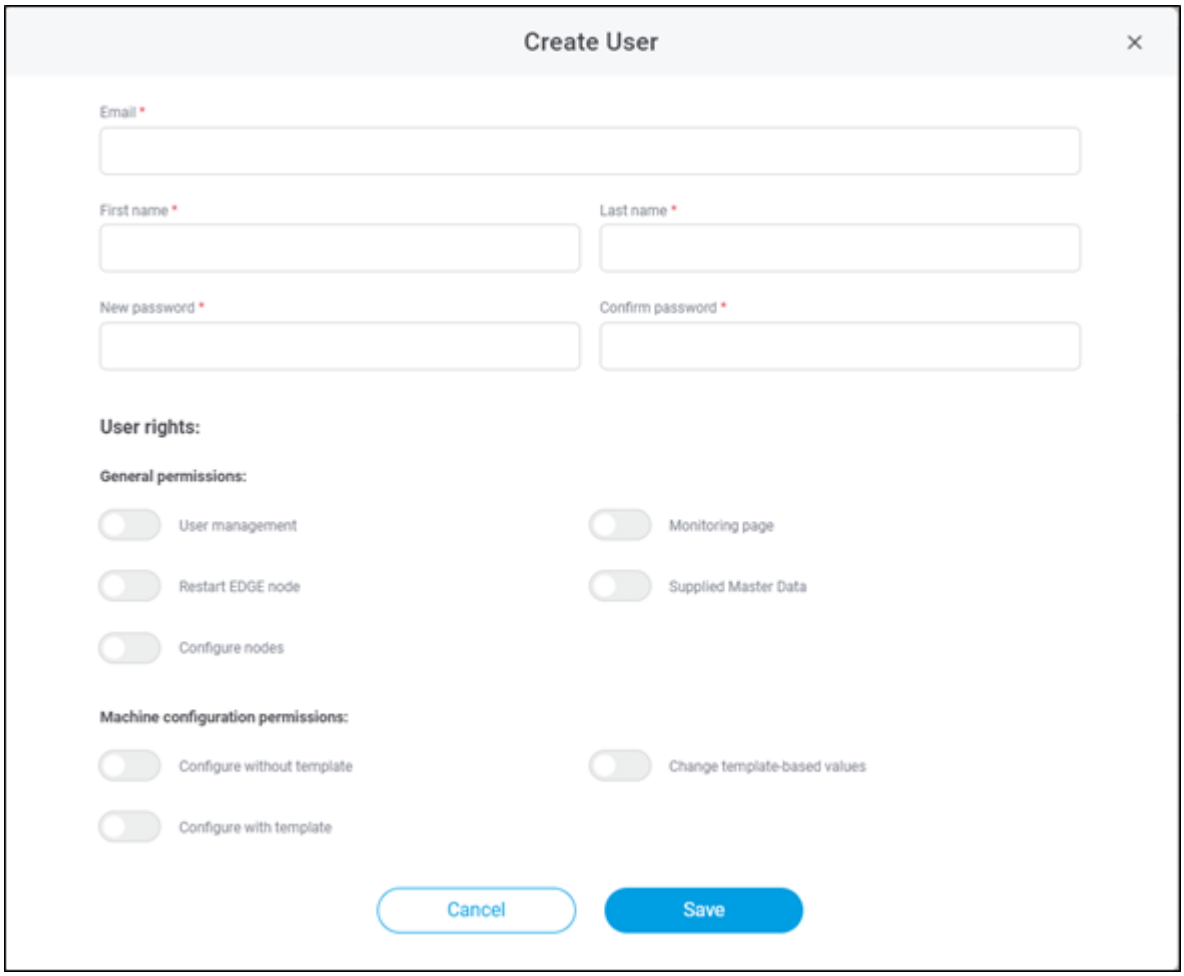


Fig. 6: Dialog for creating a new user

5.2 Supplied master data

The "Supplied Master Data" is an extension that allows asset master data to be transferred to EDGE CONNECT from third party applications.

Asset master data can be created via a third-party application such as SAP DM. To avoid having to create this again in EDGE CONNECT, the data can be transferred to EDGE Configuration via the API interface. This reduces the effort required to create an asset in EDGE CONNECT and supports consistency or synchronization of asset master data.

ⓘ “Supplied master data” can be used to create assets with basic information. Connection-specific information can be entered through a template or manually.
It is not possible to delete assets through the API. The assets can, however, be marked as deleted.

If the master data is received from the third-party application, the data can be used to create new assets in EDGE Configuration. All new assets are displayed on the **Supplied Master Data** page and are initially given the status **New Master Data**. They can be completely configured here and assigned to an EDGE node.

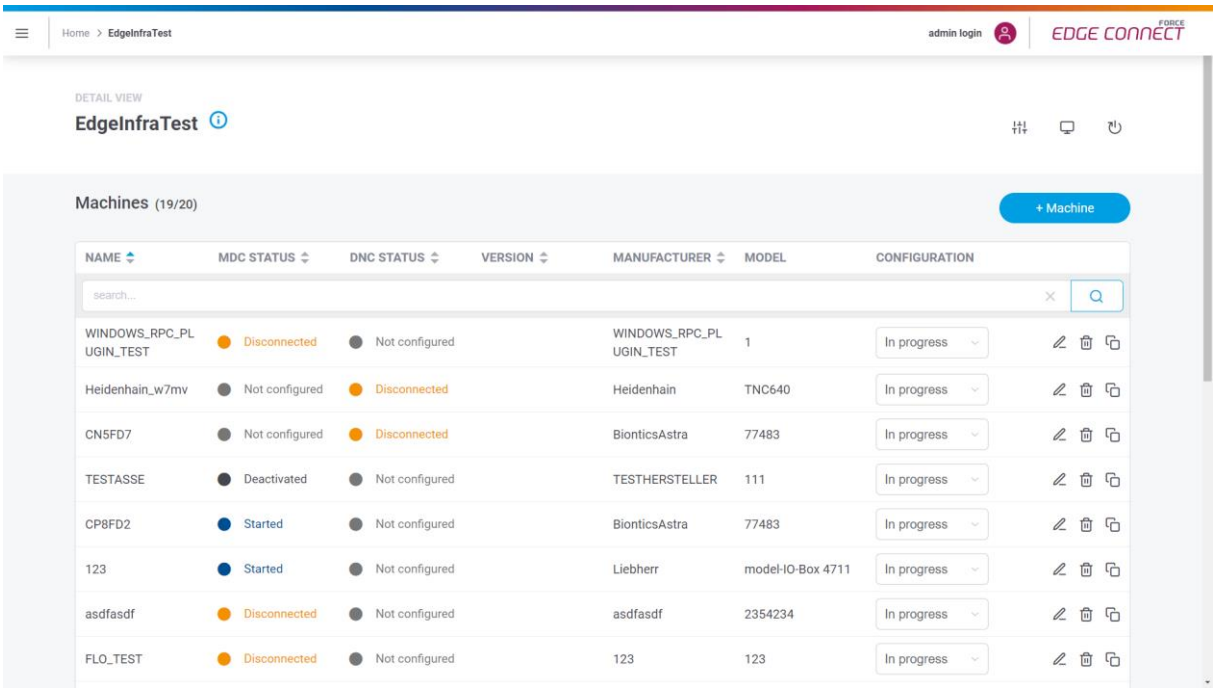


Fig. 7: List of assets with supplied master data

The following master data can be created via the API:

- Asset name
- Asset type
- Asset class
- Manufacturer
- Model
- Serial number
- External machine ID

- Inventory no.

Finishing the asset configuration

The supplied master data can be created on an existing EDGE node. For this purpose, the configuration dialog for adding an asset opens (see section 6.3). The master data received through the API get a higher priority.
Example: An asset was created via the API. Later, a template is selected for use during asset configuration. Nevertheless, the application continues to use the master data that was transferred via the API. The master data of the template will be discarded.

To provide "Supplied master data" to EDGE CONNECT:

1. Call Swagger.
   IP address (of the EDGE Configuration) + 60066/api/configuration/supplied-master-data
2. Configure and send the POST in **Supplied Master Data**.
   ➔ Asset can be found under **Supplied Master Data** in the EDGE CONNECT menu.

To create new master data on a node:

1. Call the **Supplied Master Data** function from the menu.
2. Click on the plus icon on the right of the desired master data.
3. In the subsequent dialog, select an EDGE node on which the master data is to be created.
4. Click on **Select**.
   ➔ The configuration dialog for adding an asset opens. Master data received through the API is pre-filled.
5. Make further configurations as desired (see section 6.3).

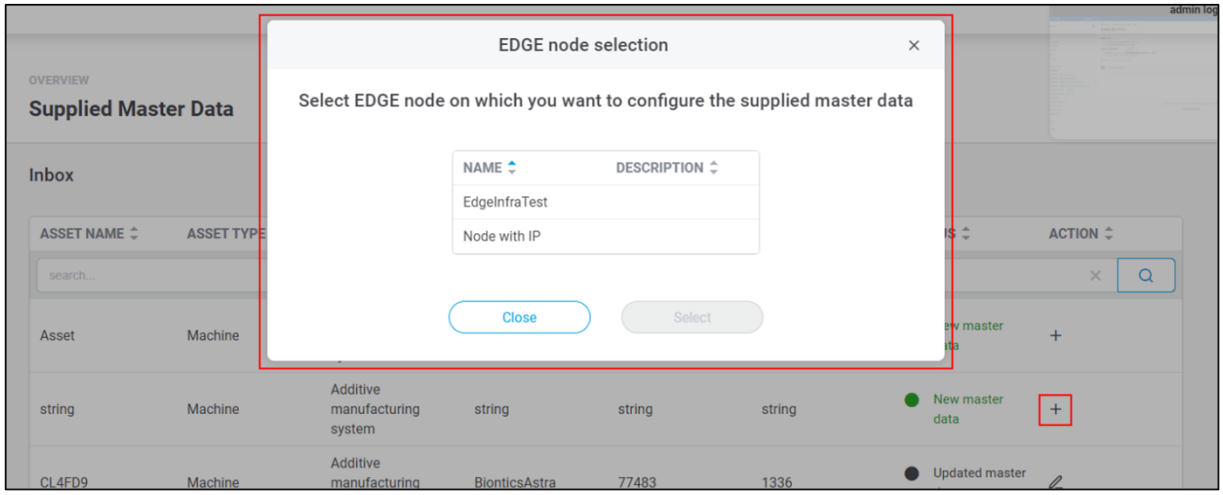


Fig. 8: Selection of an EDGE node for the creation of supplied master data

Changing asset master data via API

All asset master data can be changed via the API after it has already been initially sent. If the third-party application adjusts the master data, the status in the table of supplied master data changes to **Modified master data**. In addition, a message on the start page informs the EDGE CONNECT user that master data has been changed.
The pencil icon can be used to decide which of the changes should actually be applied.

To apply changes to master data:

1. Click the pencil icon (on the right) of the desired master data.

➔ The subsequent dialog lists all changes that have been made to the selected master data.

2. Deactivate the check box behind the desired data whose changes are *not* to be applied. By default, all switches are activated.
3. Click on **Accept**.
4. Optional: Restart the corresponding node.

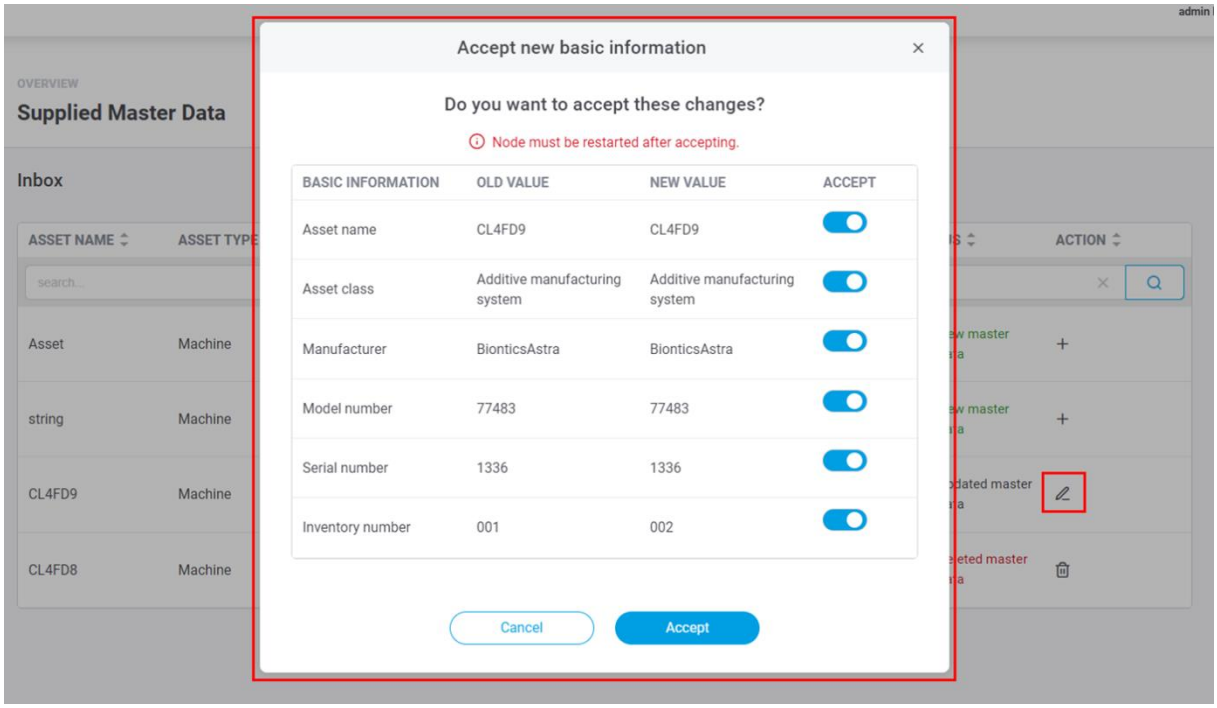


Fig. 9: Confirmation of asset master data changes

Marking an asset for deletion via API

When an asset is deleted in the third-party application, it receives the status **To be deleted** under the **Supplied Master Data** page. Once you confirm the deletion in the EDGE node, the asset is removed from the node as well as from the table of the page.

5.3 Licensing

Licenses can be imported and viewed under **Licensing**.

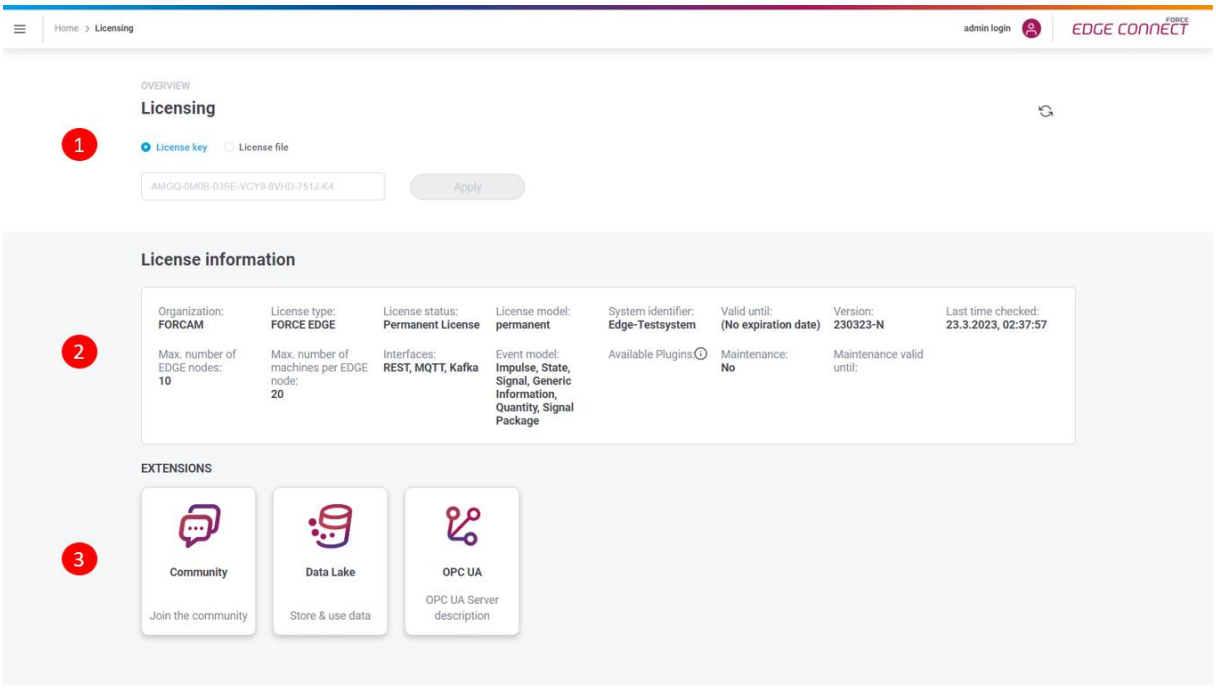


Fig. 10: Licensing and overview

(1) A new license can be uploaded as a file or entered directly as a key.
(2) License information consists of type and status of the license, number of licensed nodes and assets, maintenance, validity, and other data.
(3) All booked add-ons are listed here.
Clicking an add-on tile displays further information, such as provided URLs, for example.

5.4 Download area

The current EDGE CONNECT documentation can be downloaded in several languages from the **General** tab. Currently, the user manual and a product description are available. The manual is the document at hand, with detailed configuration instructions. The product description is a shorter document describing only the function and benefits of the application and a listing of the scope of performance functions.

In the **MDC PLUG-INS** and **DNC Plug-ins** tabs, FORCAM provides additional applications. They are needed to communicate with an asset via the corresponding plug-in. The applications enable a bidirectional communication.

5.5 Monitoring of the EDGE application

The monitoring in the menu is used to monitor the EDGE Configuration. The monitoring of the EDGE Node components is displayed on a separate page and described in chapter 7. However, the structure of the monitoring tiles is the same.

The following tile monitors the status of the transmissions of templates via the API. It specifies all the information that is logged during the process.

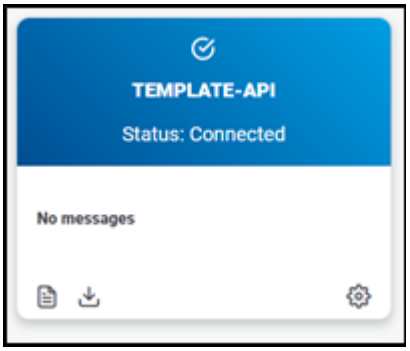


Fig. 11: Monitoring template transmissions via the API

5.6 Table Sorting

Most pages in EDGE CONNECT display data in the form of tables. You can sort the columns alphabetically in ascending or descending order.

Ascending

Descending

No sorting

Fig. 12: Alphabetical sorting of columns

Those columns that relate specifically to DNC and MDC, specify a status instead of a string. The sorting arranges the statuses alphabetically and additionally groups them by content.

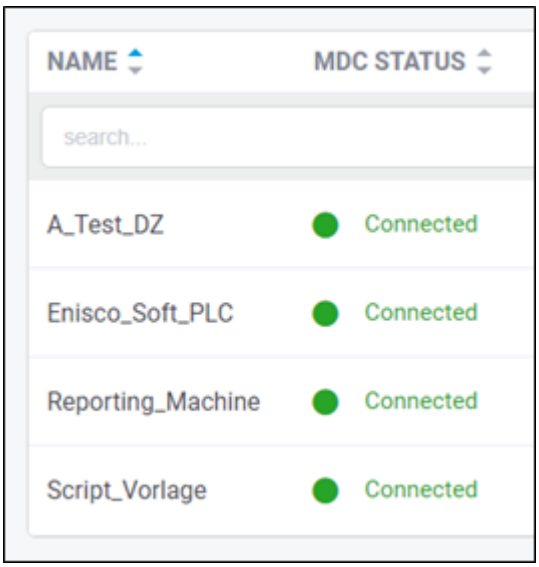


Fig. 13: Alphabetical sorting and content grouping

6 EDGE Configuration

The configuration of an EDGE node as well as an asset is done completely in the EDGE Configuration component of EDGE CONNECT. The user-friendly interface will guide you through all relevant settings and shows all nodes and the statuses in the overview.

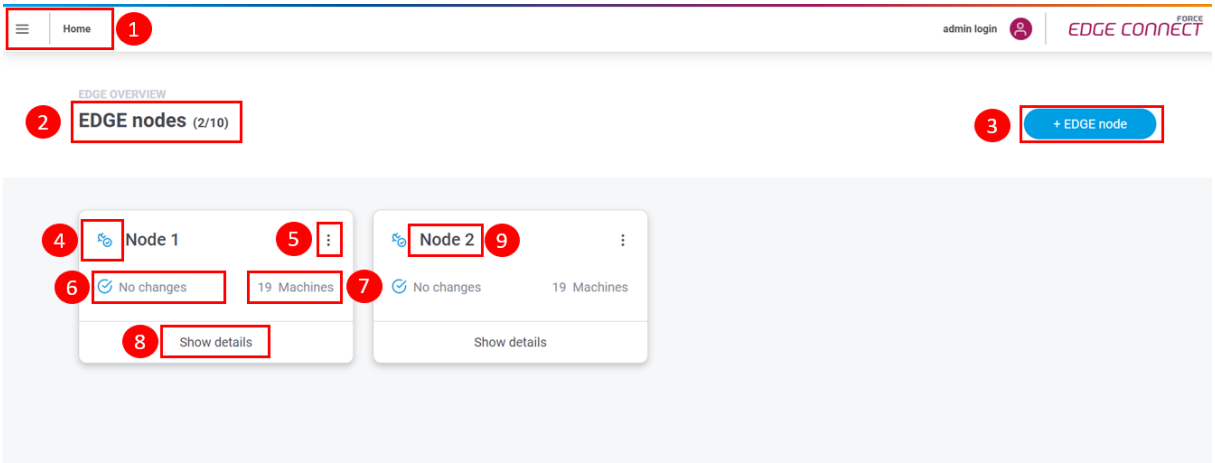


Fig. 14: EDGE CONNECT entry and overview page

(1) EDGE CONNECT Home menu
(2) Indicates the number of already configured EDGE nodes (first number) and the total number of nodes that can be configured in accordance with the license (second number).
(3) Adds a new EDGE node
(4) Status of the EDGE node
(5) Node settings menu:
- Edit
- Delete

(6) Change display of the EDGE node; indicates that the note must be restarted (if required)
(7) Number of connected assets
(8) More detailed node information:
- List of all connected assets and their status
- Option to add a new asset
- Monitoring of connected assets
- Northbound Configuration
- Restart of the node

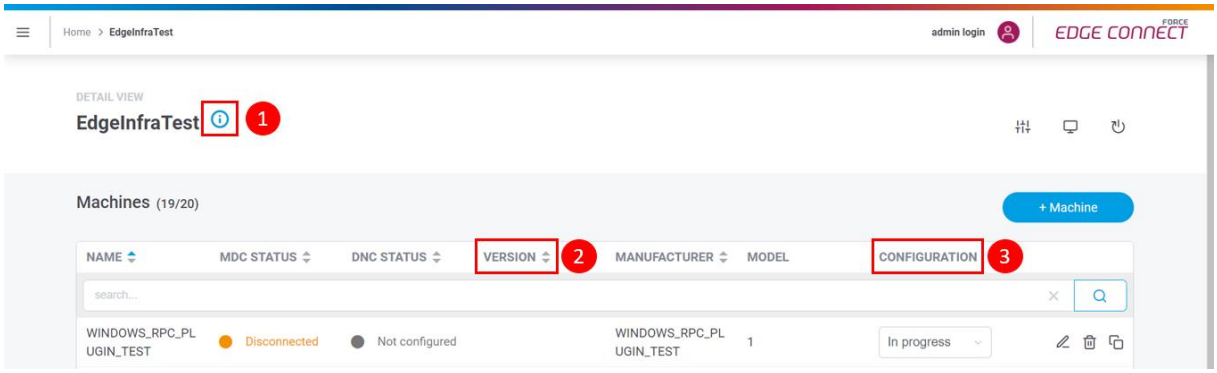


Fig.15: Asset overview as next page after clicking the EDGE Node

The **icon** (1) can be used to display additional information about the node.

The **version** (2) indicates the latest implementation of the template.

Configuration lets you manually determine a status for the configuration, to provide a better overview to users:

- **In progress**:
  The configuration is not yet complete and is to be continued at another time.
- **In validation**:
  The configuration of the asset should be checked for errors and consistency.
- **Completed**:
  Configuration is finished. This is the only status in which the MR learning cycle can take place to generate a template from the configuration.

6.1 Add EDGE node

EDGE CONNECT lets you add nodes in just a few steps. An EDGE node can represent a plant or a production line in a plant. There can be several nodes per plant. They are logically bundled so the asset workload can be distributed efficiently.

ⓘ If a configured EDGE node is removed from the interface, its configuration is preserved. If the node is recreated under the same data, it automatically adopts the previously configured data.

Fig. 16: Dialog for adding a new node

To add a new EDGE node:

1. In the node overview (Home), click on **+ EDGE node**.
2. Fill in all mandatory fields (*) in the next dialog:
   - **Name**:
     Appears as the node title in the node overview
   - **URL**:
     Consists of http + IP address+ port 60067 (Ex.: http://127.0.0.1:60067)
     Only one EDGE node can be created per URL. Other configurations are included in this restriction.
   - **API key**:
     Password that was assigned during the initial node installation
3. Optional: Add **description**.
4. **Save**.

6.2 Edit the Data Lake of a node

It is possible, of course, to edit a node after creation. In addition to the fields already mentioned, the settings for editing a node also include information and settings for the Data Lake.

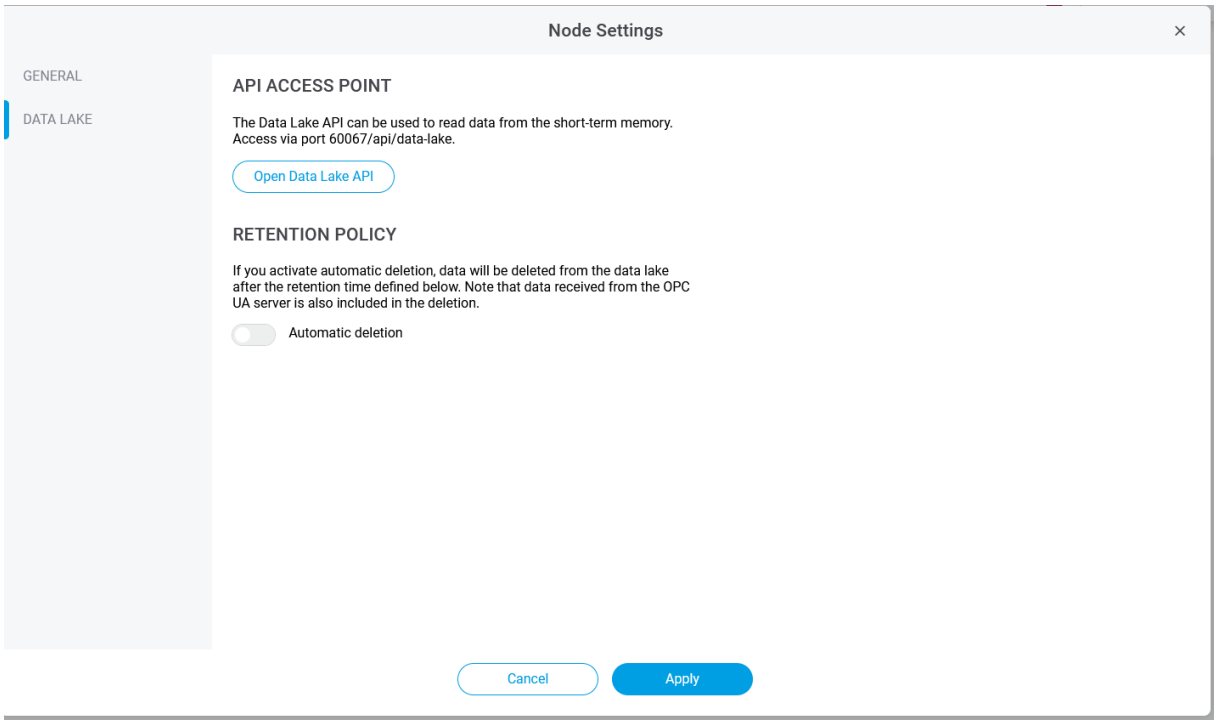


Fig. 17: Dialog for editing an already existing node

To edit an existing EDGE node:

1. Navigate to the node overview (Home).
2. Click on the three dots next to the name of the node you want to edit.
3. Click the **Settings** option.
4. Navigate to the **DATA LAKE** tab on the left.
5. Edit the values.
6. Click **Apply.**

While editing an existing node, the retention policy for the Data Lake can be defined. A retention policy describes a rule that defines how data is treated within the Data Lake. This only refers to the deletion of asset-generated data (configuration data or other is not affected). A time frame can be defined for which the asset-generated data will be stored in the Data Lake. The time is defined in days. One retention policy refers to exactly one EDGE node and must therefore be created/defined separately for each node, as the retention policy is not activated by default. Once a retention policy has been created, it is automatically active and takes effect, i.e., “outdated” data is deleted. This process cannot be stopped, but the time frame of an existing policy can be changed afterwards. This is not the same as to cancel the process, because the application has already started processing the initial policy, so data might already have been deleted.

⚠ The Data Lake should be considered a short-term repository. The technical requirements from the System Requirements document must be observed!

6.3 Add asset

A Configuration Wizard guides you through eight steps required to connect an asset. This is where MDC/DNC controls are configured and asset signals are defined, among other things.

ⓘ Negative values are not permitted in the asset configuration.

Once a step is completed, it is highlighted in blue in the top bar.
To return to an already completed step, click on the step.
While an already configured asset is edited, each configuration page can be selected and called up directly.

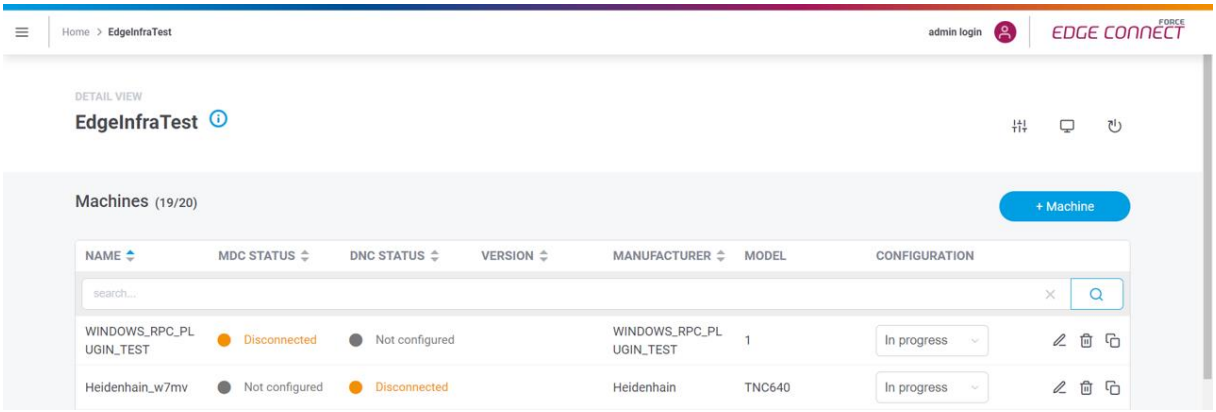


Fig. 18: Dialog for configuring an asset in EDGE CONNECT

To add an asset:

1. Click on **+ Machine** in the node details.
   ➔ The Configuration Wizard guides you through the following eight steps for configuring an asset.
2. Click **Apply** to finish.

⚠ All settings in the Configuration Wizard must be saved at step ⑧ with **Apply**, otherwise the entire configuration will be lost.
Opening another menu will also cause the configuration to be discarded.

6.3.1 ① Select template

In EDGE CONNECT, multiple assets of the same type do not have to be completely reconfigured each time: Once an asset has been configured, it can be entered as a template in the Machine Repository and will then be offered for the next asset connection in this mask. If the template is selected at this point, all settings are automatically used for this asset and all configuration fields that are not asset-specific are pre-filled. Only the information related to the asset (e.g., serial number) and to the connection (e.g., IP address or port of the asset or controller) must still be edited.

The **VERSION** indicates the revision status of the template. If a template is revised, the version number is automatically incremented by 1, and the earlier version is overwritten. Version 0 means that no script is configured in the corresponding template.

ⓘ This step is only available if the MR extension is used. If no template is configured or MR is not in use, the Configuration Wizard will start in step ②.

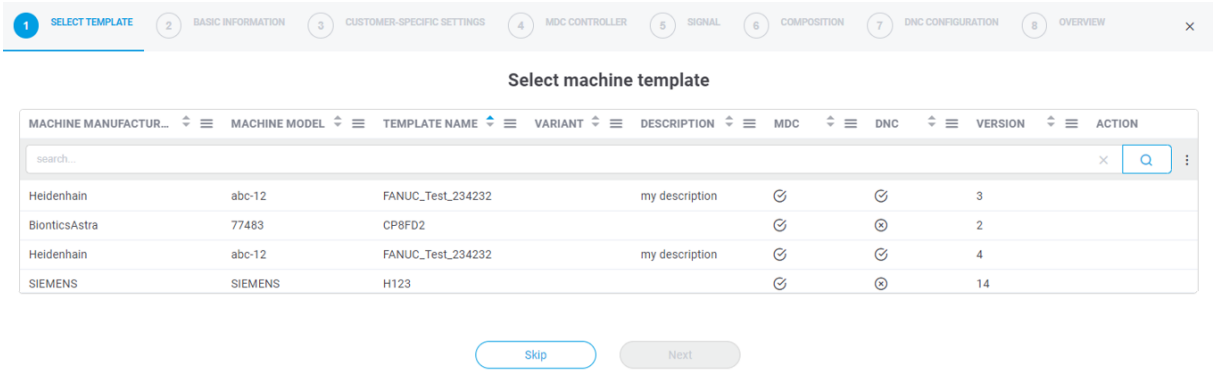


Fig. 19: Configuration Wizard – select template

1. In the list, select the desired template for connecting the asset.
2. Click **Next**.

ⓘ If you do not use a template for asset connection, click **Skip**.

6.3.2 ② Basic information

In this step, basic information of the asset to be configured is defined, such as name or serial number. In addition, you determine whether an MDC or a DNC control is to be configured, or both. With the MDC controller, signals are collected from the asset and transmitted or written to it. DNC controllers are used to transfer NC files to the asset.

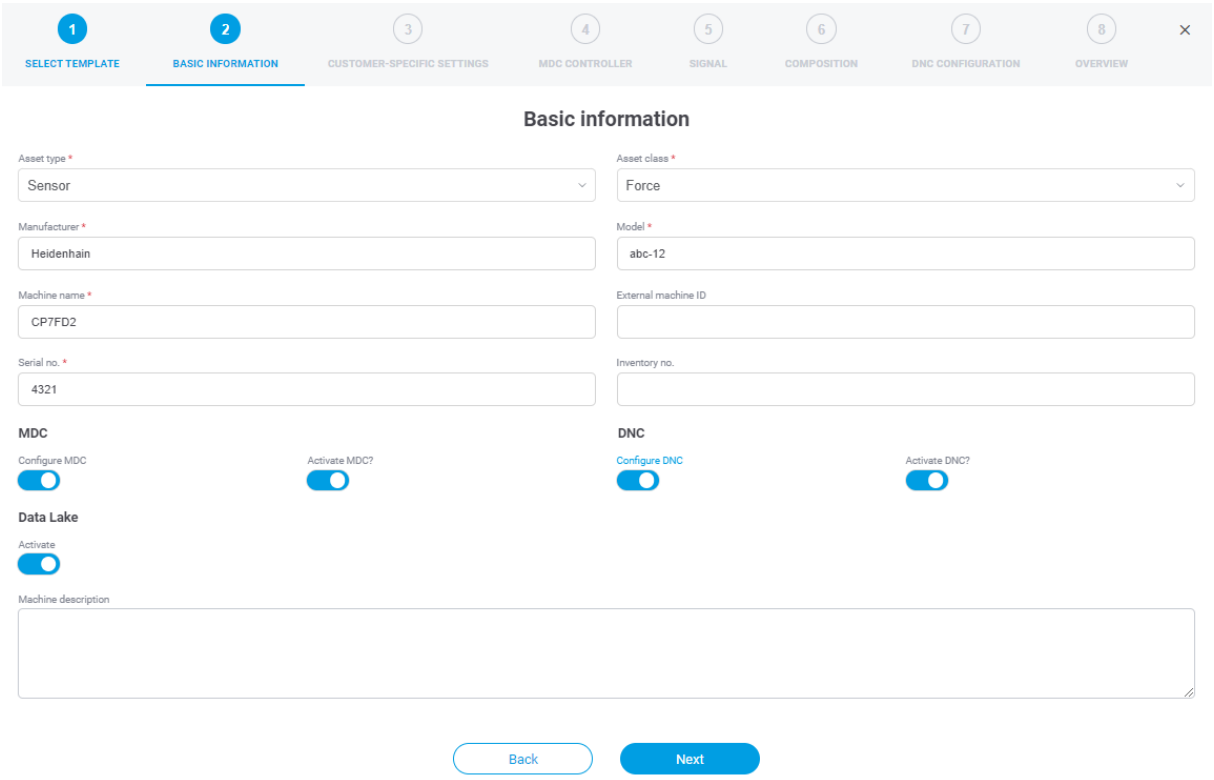


Fig. 20: Configuration Wizard – Basic Information

Input field	Description
Asset type	Selection: – Machine – Sensor – IT System
Asset class	Selection varies depending on content in the Asset type field
Manufacturer	Free input field
Model	Free input field
Machine name	Free input field
External machine ID	Free input field, optional
Serial no.	Free input field
Inventory no.	Free input field, optional

MDC **Configure MDC** **Activate MDC?**	 Activate switch if MDC is to be configured Activate switch if MDC is to be used
DNC **Configure DNC** **Activate DNC**	 Activate switch if DNC is to be configured Activate switch if DNC is to be used
Data Lake - Activate	Activate switch if Data Lake is to be used (Switch is only displayed if the Data Lake component is available)
Machine description	Free input field, optional

6.3.3 ③ Customer-specific settings

This step enables individual, plant-specific information to be added to an asset to further supplement the asset's data. This data can later be retrieved from the API to provide more information to a third-party system.

Example: Name = location, value = hall 2.
This is where an additional locational aspect is added to the asset data to help accurately locate the asset in the event of a malfunction.

ⓘ This step is optional.

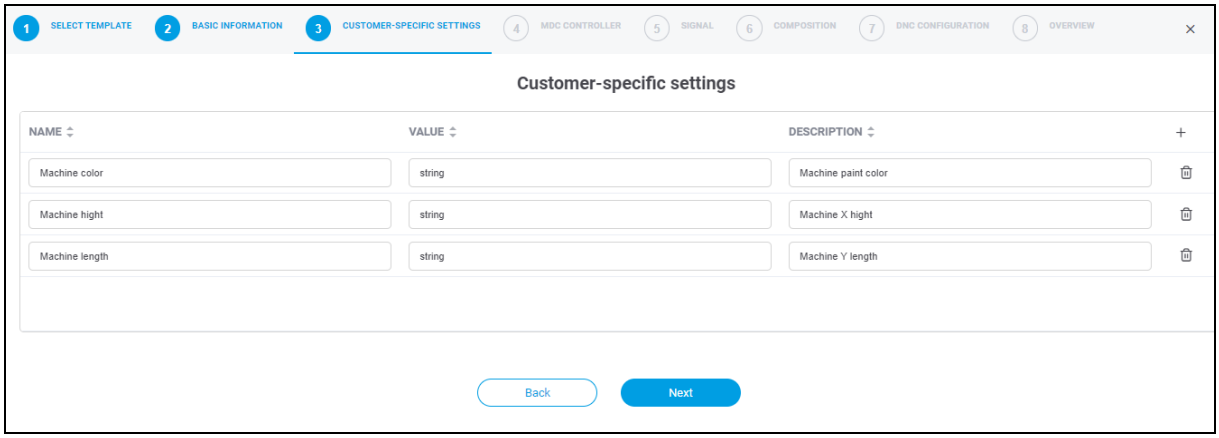


Fig. 21: Configuration Wizard – Customer-specific settings

1. Click on the **+** on.
2. Enter the necessary parameters.

6.3.4 ④ MDC Controller

ⓘ This step is only available if **Configure MDC** was selected in step ②.

In this step, an MDC control can be configured. This specifies the way the asset is to be linked to EDGE CONNECT.
Section 0 lists all FORCAM plug-ins that are currently available.

Here you select type and bus type. The other input fields vary depending on this selection. Mandatory fields are marked with an asterisk (*).
The bus type is a specific communication protocol of the controller type. Many controllers only use one protocol and thus only one bus type is available for selection.

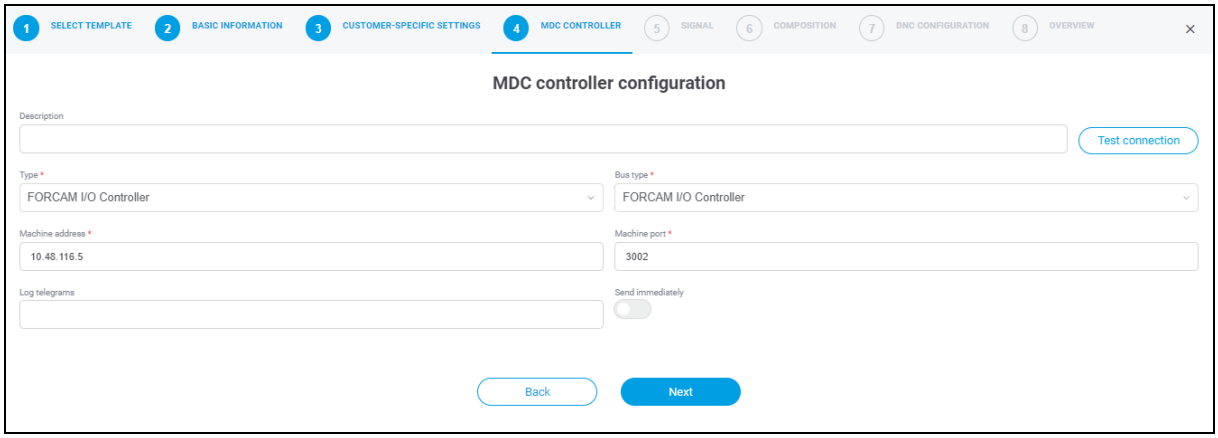


Fig. 22: Configuration Wizard – MDC controller

ⓘ The **Log telegrams** function is used to store the UDP telegrams in the MDC log. The number of characters to be logged for each telegram is specified in the input field.

6.3.5 ⑤ Signal

This step defines which signals are read from the controller. Depending on the configuration of the MDC controller (step ④), different listings of the signal types are displayed. The Data Lake can be used to record and save all data. Data storage in the Data Lake can be switched on and off for each signal. Units can be recorded on individual signals (e.g., Degrees Celsius or liters per minute), and scaling factors can be defined. The scaling factor makes it possible, for example, to infer the temperature by means of the resistance detected at an asset.

ⓘ If the **ACTIVE** switch for the signal is deactivated, it cannot be used in step ⑥ COMPOSITION.

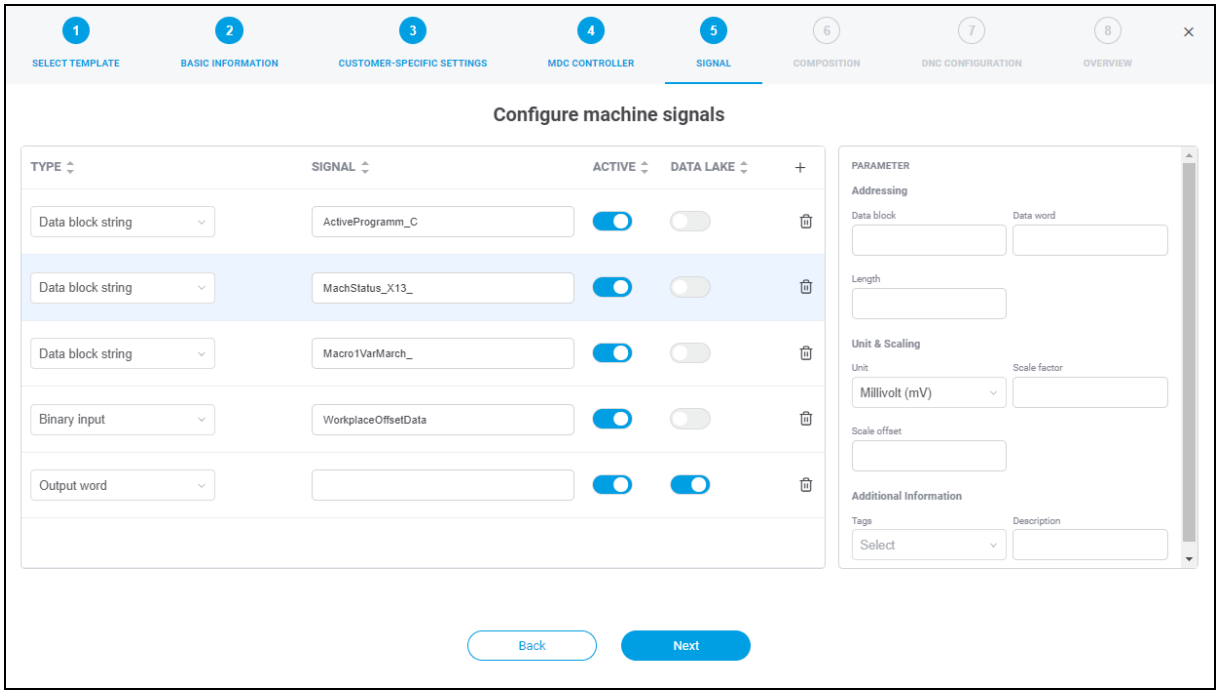


Fig. 23: Configuration Wizard – Signal

1. Click on the **+** icon.
2. Select the **TYPE** , enter a name for the **signal** and ,optionally, activate the switch for **DATA LAKE**.
3. Specify plug-in-specific signal **parameters**.
3. Optional: Enter values for **Unit & Scaling** and **Additional Information**.
4. Click **Next**.

Input field	Description
TYPE	Data type of the signal, selection differs depending on the type of asset connection
SIGNAL	Name of the signal, free input
ACTIVE	Selection whether or not the signal should be actively detected
DATA LAKE	Selection whether or not the signal should be stored in the DATA LAKE ⓘ Option is only available if the Data Lake function is enabled.
Parameter	**Description**

Input field	Description
Addressing	Address, input options depending on TYPE
Unit & Scaling	– Unit: selection from drop-down menu – Scale factor: Factor by which a value is multiplied in order to interpret it correctly within its reference system – Scale offset: Offset that must be taken into account when interpreting values
Additional information	– Tags: keywords with which the signal is enriched – Description: Free input

⚠ Once created, signals can no longer be changed.

6.3.6 ⑥ Composition

In this step, the received signals are interpreted and interpretation conditions are defined. As a result, measurement values, maintenance information and production states are available. This makes it possible to draw logical conclusions about the behavior of the asset.

ⓣ We recommend creating internal company guidelines for signal composition. This creates a uniform data model across all assets, which forms the basis for comparative evaluations.

There are two ways to implement this signal interpretation: In the SCRIPT section, text-based code is displayed and edited (see Fig. 25) see whereas in the **GRAPHIC** section graphical blocks can be used (Fig. 24).

⚠ Simultaneous editing in **SKRIPT** and **GRAPHIC** is not possible, neither can you switch from **SKRIPT** to **GRAPHIC**.
Once changes are made in the script editor, further editing must be done there, and the code will no longer be displayed under **GRAPHICS**.

Graphical editor

The blocks in the graphical editor are programming blocks/modules that can be put together and connected, similar to the individual pieces of a puzzle. The advantage of this modular system is that you can create the required commands even if you are new to programming in general.

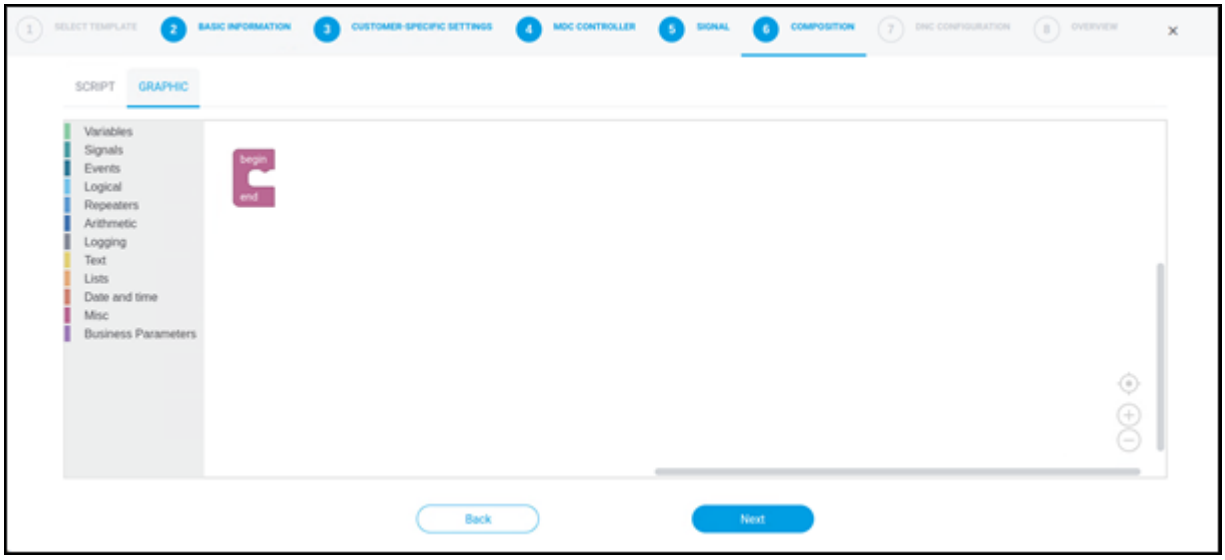


Fig. 24: Graphical editor

On the left side of the screen, all available function categories are listed, divided, and sorted by color. Drag-and-drop can be used to move the required blocks to the editing area on the right and place them in the correct order. This is where the actual asset logic is defined.

ⓣ For a detailed description of the individual function categories of the blocks, see the **Graphical Composition** manual.

Script editor

The annex of this manual contains sample scripts and script functions (see sections 7.5 and 7.6).

⚠ Only users that are familiar with programming should work in the **SCRIPT** editor.

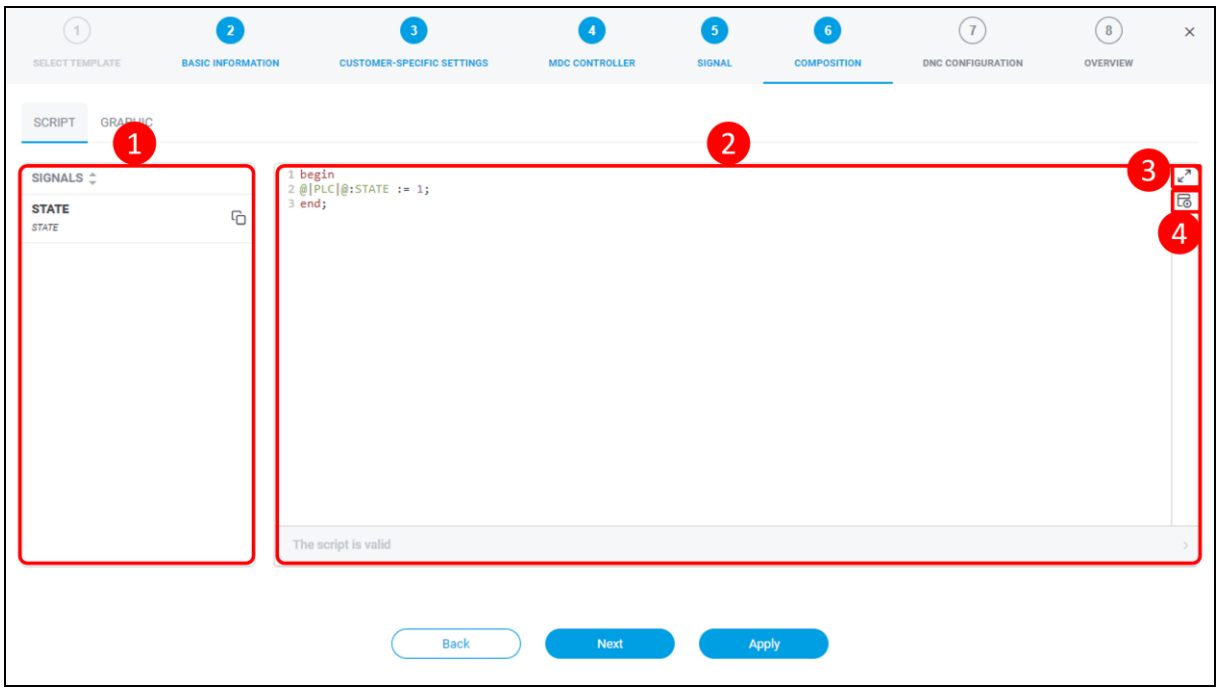


Fig. 25: Script editor

(1) Shows the signals that were added in step 5 of the Configuration Wizard
(2) Editing area with current script
(3) Zoom in/out view (full screen)
(4) Check the current script for validity

ⓘ The script must be error-free. You can only proceed to the next configuration step if the script has no errors.

6.3.7 ⑦ DNC Configuration

In this step, a DNC control can be configured. Specifies the way an NC file is to be transferred to the asset.
Section 0 lists all FORCAM plug-ins that are currently available.

ⓘ This step is only available if **Configure DNC** was selected in step ②.

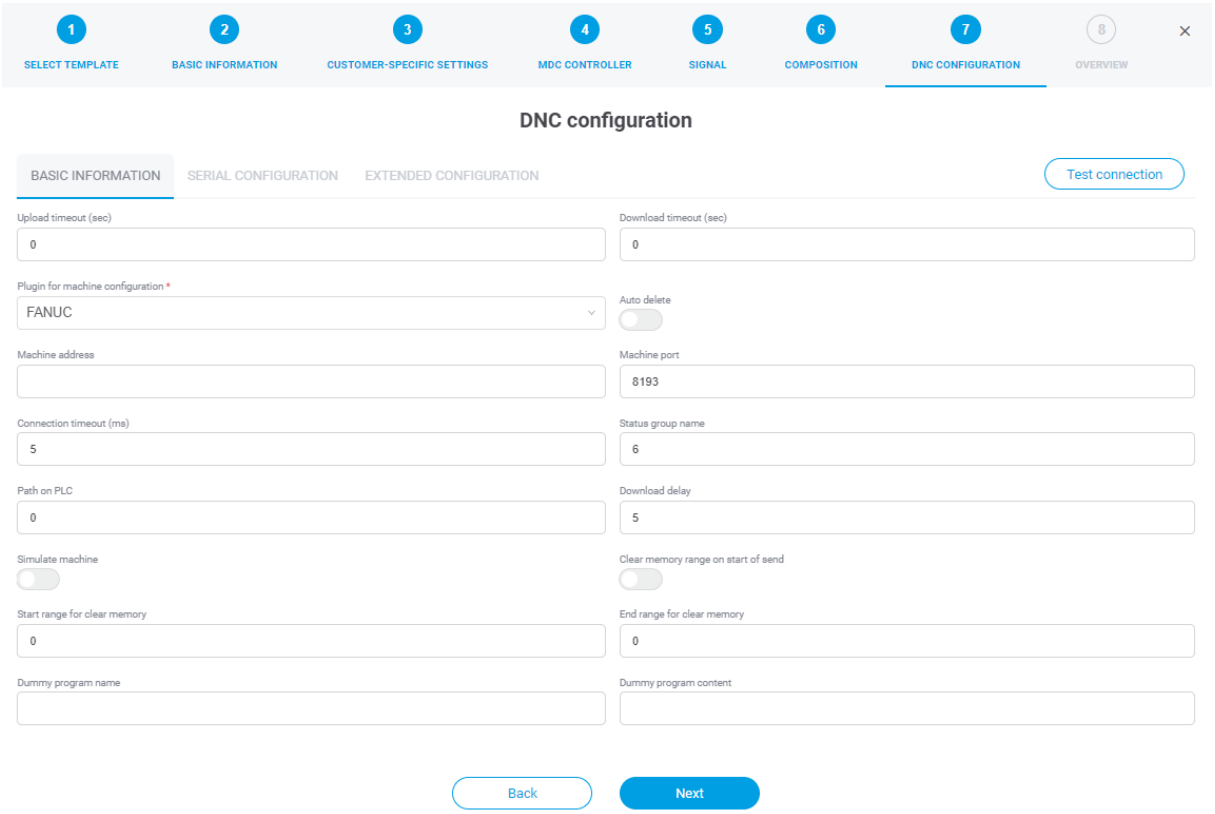


Fig. 26: Configuration Wizard – DNC configuration

Input field	Description
Upload timeout (sec)	Specification in seconds
Download timeout (sec)	Specification in seconds
Plug-in for machine configuration	Drop-down menu containing all available plug-ins The other input fields are displayed depending on this selection. Mandatory fields are marked with an asterisk (*).
Auto delete	Switch activated = NC file is automatically deleted from the asset after reading.

ⓘ Once the DNC configuration is completed, the connection can be tested.

6.3.8 ⑧ Overview

This step provides a summary of the configuration settings from all steps and lists all defined signals. After confirming, the asset is mapped with the specified configuration and is therefore digitized. The configured asset appears under the specified name in the overview (see Fig.15:).

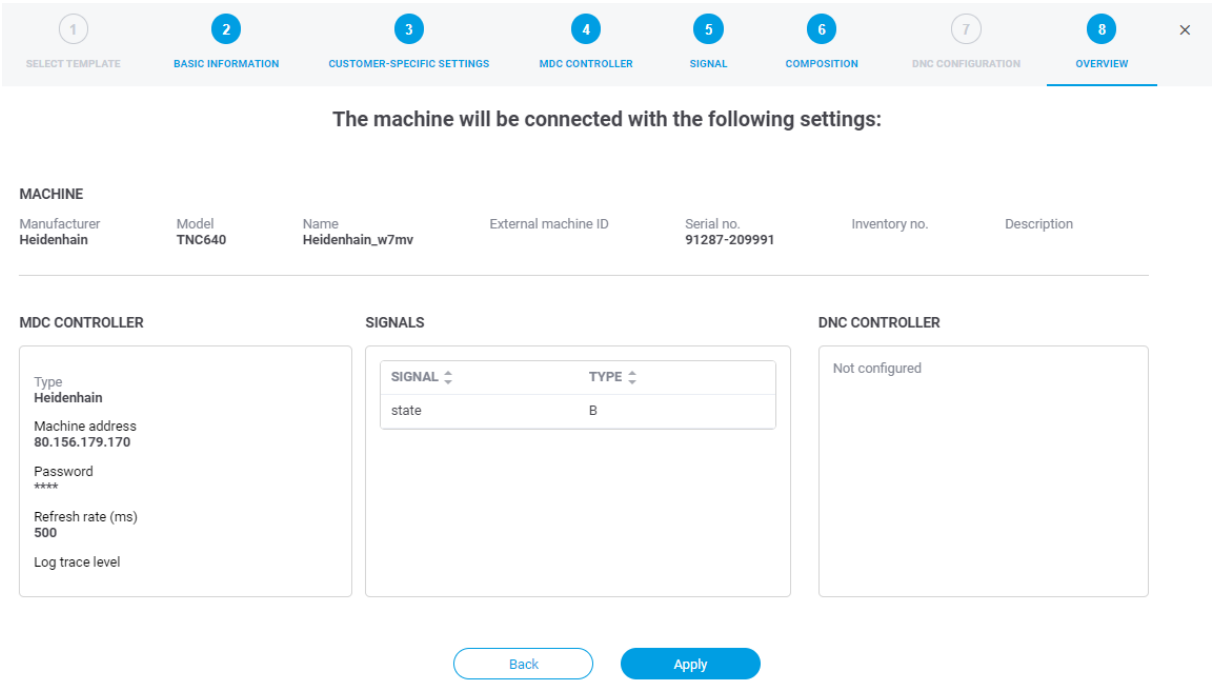


Fig. 27: Configuration Wizard – Overview

6.4 Northbound Configuration

The Northbound configuration specifies how the signals are sent to a superordinate system. Payload and endpoint are predefined by default, but they can be customized.

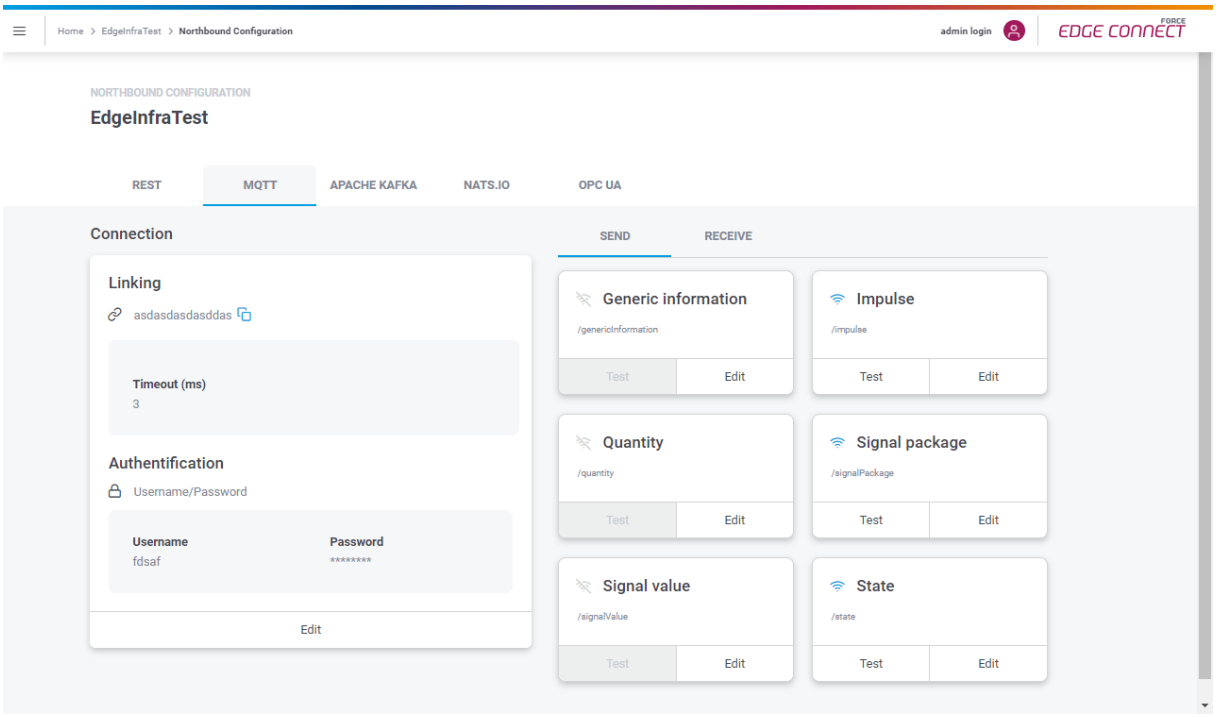


Fig. 28: Northbound configuration in EDGE CONNECT

Events are outgoing events that are generated via the script. For this, there are script functions available that generate a corresponding event depending on the type.
There is a standardized Event for each type of event. For example, the **Quantity** event type sends the quantity produced by the asset. All available events are listed in section 8.4.

The **Payload** section in the JSON body defines how the message to the superordinate system should look like. Finally, the placeholders (wildcards) are replaced by the corresponding existing signals. Example of an event structure:

```
{
        machineId: $machineId$
        machineName: $machineName$
        externalMachineId: $externalMachineId$
        reference: $reference$
        timeStamp: $currentUTCTimeStamp$
        signalName: $signalName$
        value: $value$
        unit: $unit$
}
```

Wildcards can be used to edit event structures, which can pass on different types of information. This can be used, for example, to convert the machine ID or the time stamp to UTC. Chapter 8.6 lists and explains all available script functions.
If the **Active** switch is enabled, the corresponding event will be sent. Events that are not enabled will not be sent.
An enabled event can also be tested by clicking **Test**. In the subsequent dialog, values such as machineId (machine ID) or value can be entered to generate and execute the signal as an example without influencing the actual asset connection. This allows events to be tested in advance without having to execute them in the live environment.

6.4.1 Signals and events from EDGE to superordinate system

There are four technical options for supplying signals and events from an EDGE node to a third-party application.

ⓘ The supply can be configured in the EDGE node itself.

HTTP/REST

To supply the external system, any REST endpoint provided there can be used. The HTTP methods POST and PUT are supported.
The following standards are implemented as HTTP authentication methods:

- Basic Authentification: Authentication according to RFC 2617 by entering the user name and password **(see https://datatracker.ietf.org/doc/html/rfc2617)**.
- Client credential flow: Authentication according to OAuth 2.0 RFC 6749 via client ID and client secret known to the system. (See **https://auth0.com/docs/flows/client-credentials-flow**). This type of authentication is performed without user intervention, i.e., in the background.

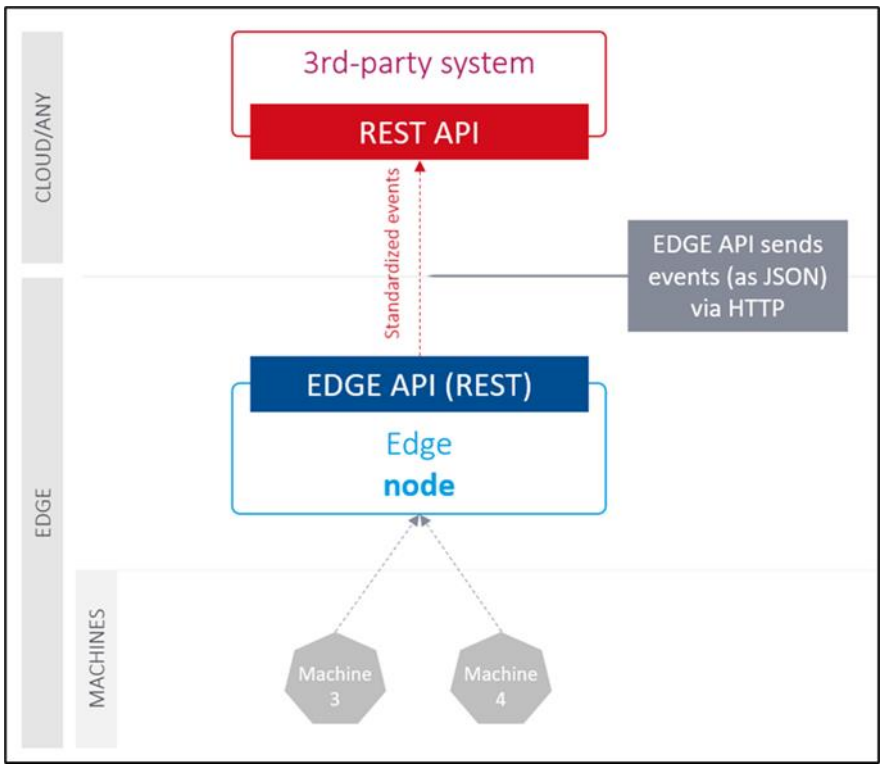


Fig. 29: Communication with superordinate systems via HTTP/REST

MQTT messaging

Any MQTT broker be served, if provided by the customer or partner.

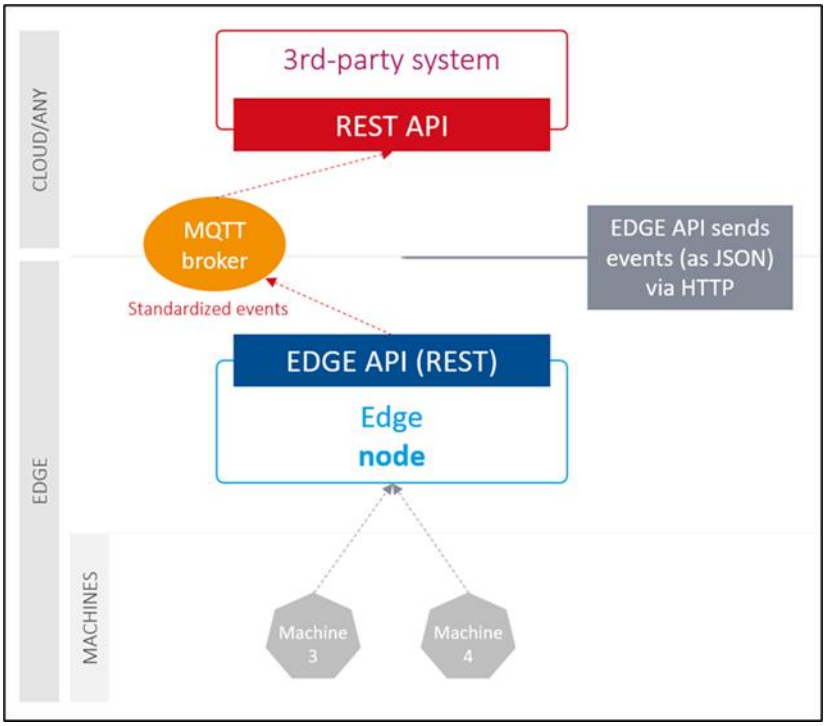


Fig. 30: Communication with superordinate systems via MQTT broker

Apache Kafka

The third-party system can be supplied using Apache Kafka, if provided by the customer or partner.

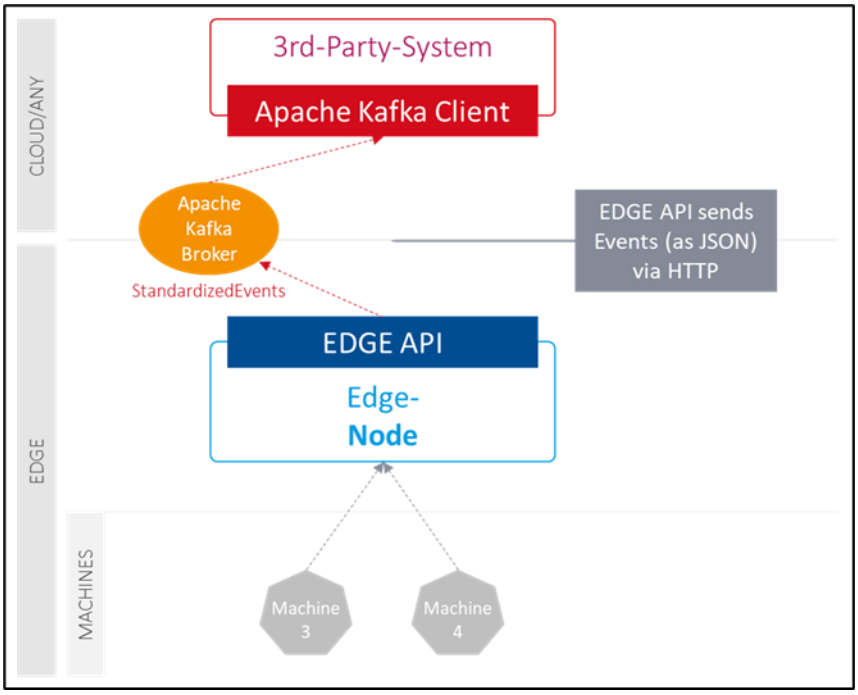


Fig. 31: Communication with superordinate systems via Apache Kafka

OPC UA

FORCAM provides an OPC UA server with “Data Access” functionality. This extension makes various asset data available via the defined OPC UA interface. The information models are prepared dynamically based on the existing assets, configured in the EDGE node.
The user can connect to the server via the specified URL to retrieve the desired data. We assume that the client required for data retrieval already exists.
It is not only possible to retrieve the current values of an event or signal, but also the history. To be able to process these historical data sets, EDGE CONNECT supports the Historian functionality of OPC UA (only available if Data Lake is used). The Historian acts as a data logger with SQL databases. It logs historical data and can additionally be used as a gateway to access real-time data from all underlying OPC UA servers.
In EDGE CONNECT, the user has the option to assign login credentials. This data must then be entered correctly when connecting to the server via the client.

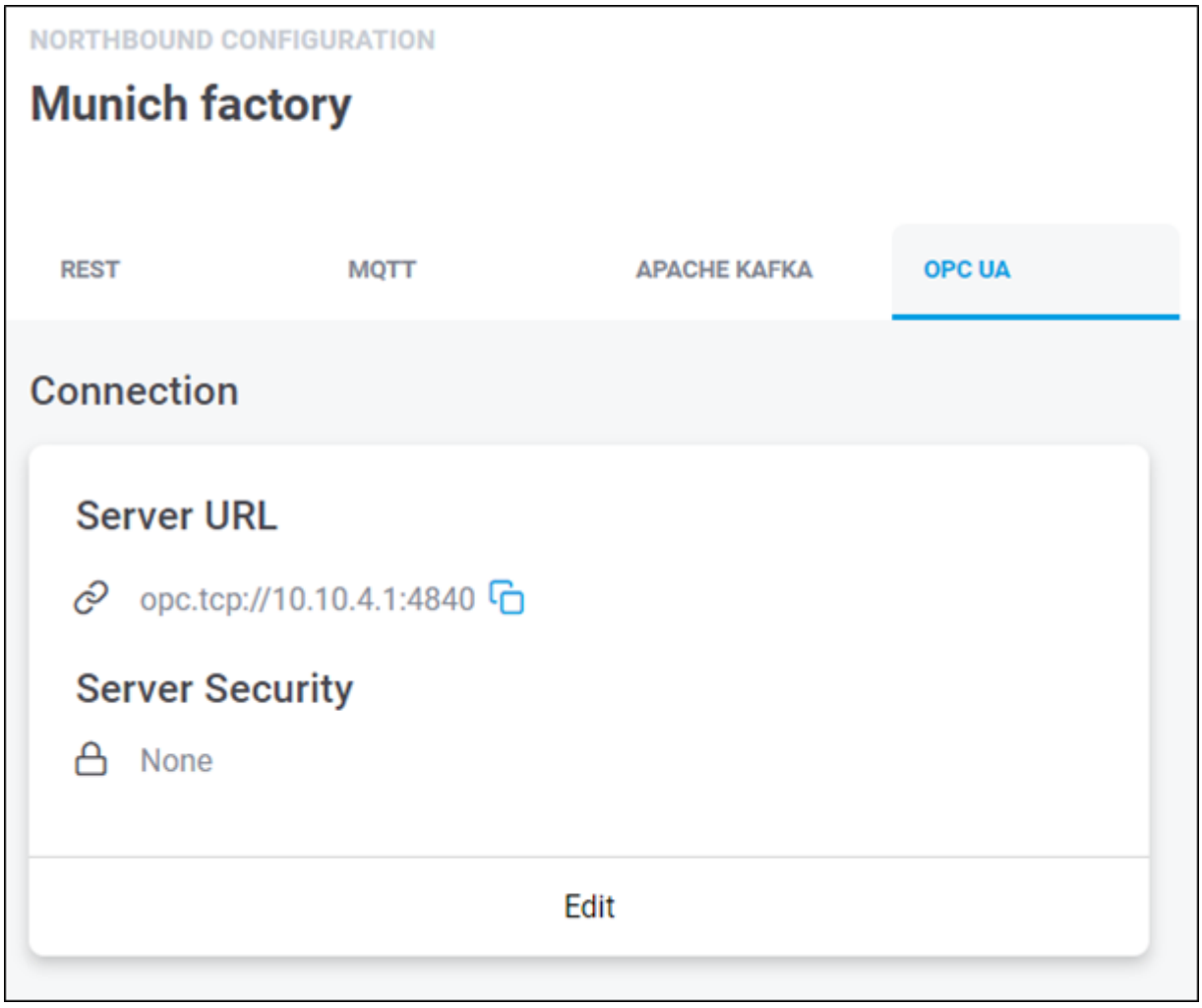


Fig. 32: Event Configuration via OPC UA

NATS.io

Connection to a NATS infrastructure EDGE CONNECT is also possible for sending northbound-related information. The NATS interface does not support the reception of information (e.g., business parameters).

Core NATS as well as NATS JetStream can be used for transmitting events. In both cases, different subjects and streams can be defined in order to facilitate data distribution.
Placeholders make it possible for the user to freely configure the content of the events to be transmitted and adapt it to the target system.

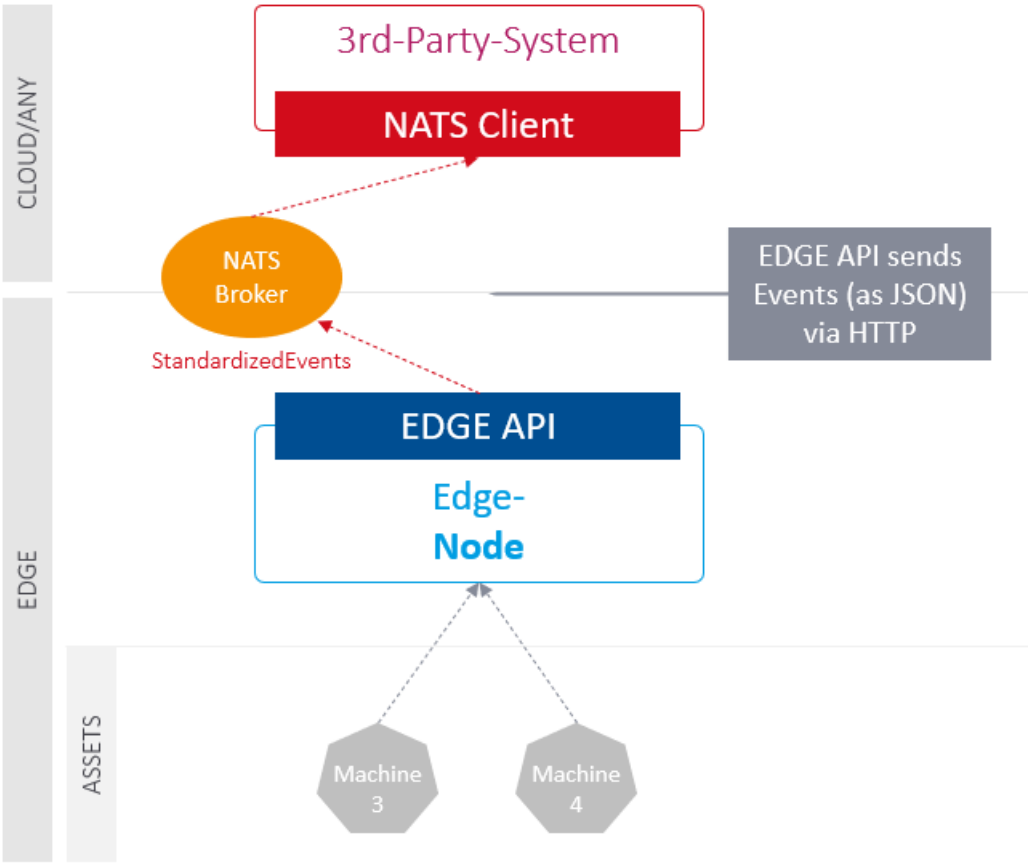


Fig. 33: Communication with superordinate systems via NATS.io

6.4.2 Data & documents from superordinate system to EDGE

EDGE CONNECT can be supplied with data and documents via the EDGE API.
The transmission of NC files is technically possible via HTTP/REST. Writing business parameters and signal values is also possible via MQTT in addition to HTTP/REST.
The following interfaces are provided:

Function	Description
Transfer of process and reference parameters	These business parameters can be used in Signal Composition to supplement standardized events (e.g. order number or cycle time), among others.
Transfer of signal parameters	Parameter values for specific signals can be transferred. These are written directly to the asset control.
Transfer of documents	NC programs can be transferred to the asset.

Table 2: Interfaces for transferring data and documents

6.4.3 Configuring an event

Before events can be created, first the connection must be determined through which to communicate with the asset (interface between the asset and the third-party system).
Then, depending on the interface, standardized events can be generated.

To determine the connection:

1. Click on the Northbound Configuration icon in the upper right part of the details view of a configured asset (see Fig.15:).
2. In the subsequent screen, select the tab for the interface you want to use.
   - REST
   - MQTT
   - Apache Kafka
   - NATS.io
   - OPC UA
3. Under Connection, click **Edit**.
4. Enter the connection information in the subsequent dialog.
   (See Table for the information that is necessary for each interface.)
5. For REST and Apache Kafka only :
   Enable the **Check SSL Certificate** switch when communicating via a standard security certificate.

ⓘ Disable the switch if the application connection is set up without a valid or with an unsigned security certificate. (For example, for self-created certificates).

6. Select desired authentication and enter login data.
7. **Save**.

Interface configuration

Configuration	Description
REST	
URL*	Text field to enter the server URL
Content type	Data format of the message content
Number of connections*	Number of active, parallel connections to the external system
Timeout (ms)*	Limits the time for connection attempts to the server
Check SSL certificate	Automatically checks the SSL certificate If the certificate was created by the user, the switch should be disabled.
Authentification	None: No authentication HTTP BasicAuth: User / Password Auth 2.0 Client credential flow: Authentication via defined standard
MQTT	
URL*	Text field to enter the server URL
Timeout (ms)	Limits the time for connection attempts to the server
Authentification	None: No authentication Username/Password: Authentication via user credentials
User*	Text field for the user
Password	Text field for the password
Apache Kafka	
Bootstrap Servers*	Text field to enter the server URL
Client ID	
Check SSL certificate	Automatically checks the SSL certificate If the certificate was created by the user, the switch should be disabled.
Security Protocol	Plain Text SASL Plain Text SASL SSL
NATS.io	
Server URL*	Text field to enter the server URL
Timeout (ms)*	Limits the time for connection attempts to the server

Configuration	Description
Reconnect attempt after (ms)	Time to wait until the next attempt to reconnect to the server
Activate TLS	Activates SSL/TSL encryption
Server Security	None: No authentication Username/Password: Authentication via user credentials JWT/NKey Credentials: Authentication via certificate file
Streaming behavior	Core NATS: Sent files are not saved. JetStream: Sent files are saved. Thereby applies that the message is transmitted at least once (Quality of Service Level 1).
OPC UA	
Server security	None: No user password required Login information: Username / Password required

*Mandatory field

Table 3: Interface configuration of the Northbound Configuration

To configure an event:

ⓘ Depending on the selected interface, certain evens can be configured. The dialog windows therefore look somewhat different.

1. Under Event, click **Edit**.
2. Configure the events as desired.
   Copy placeholder on the left and paste it into the area on the right.
   Or
   Write directly into the right area.
   ⓘ Script language can be switched between JSON, XML or text (see Fig. 33)
3. Activate the event with the **Active** button.
   An active event is indicated by a blue wave icon in the Northbound Configuration overview.
   For inactive events, the wave icon is gray and crossed out.
4. **Save**.

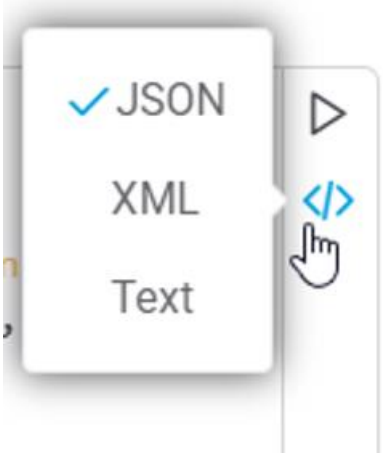


Fig. 34: Change script language

6.5 Integration

The following API interfaces are available for EDGE CONNECT:

EDGE Node	
EDGE Node API	IP-Adresse + 60067/api/management
Data Lake API	IP-Adresse + 60067/api/data-lake
EDGE Configuration	
EDGE Configuration API	IP-Adresse + 60066/api/configuration
Monitoring API	IP-Adresse + 60066/api/monitoring
Literals	IP-Adresse + 60066/api/literals

Authentication in Swagger

Authentication for EDGE Node API and Data Lake API:

1. Make the following entries under **Basic Authentication (http, Basic)** :
   - **Username**: *usertest@mail.com*
   - **Password**: *secret* (API key is assigned during installation)
2. Click **Authorize**.

Authentication for EDGE Configuration:

3. Under **Basic Authentication (http, Basic)** enter the **user name** and **password** of the respective user.
4. Click **Authorize**.

7 Monitoring

EDGE CONNECT provides the option to monitor the individual components of the software via the Monitoring page. The page indicates whether a component is running without errors or if there are any malfunctions. The monitoring can be called up via the Monitoring icon in the upper right area in the asset overview (see Fig.15:).

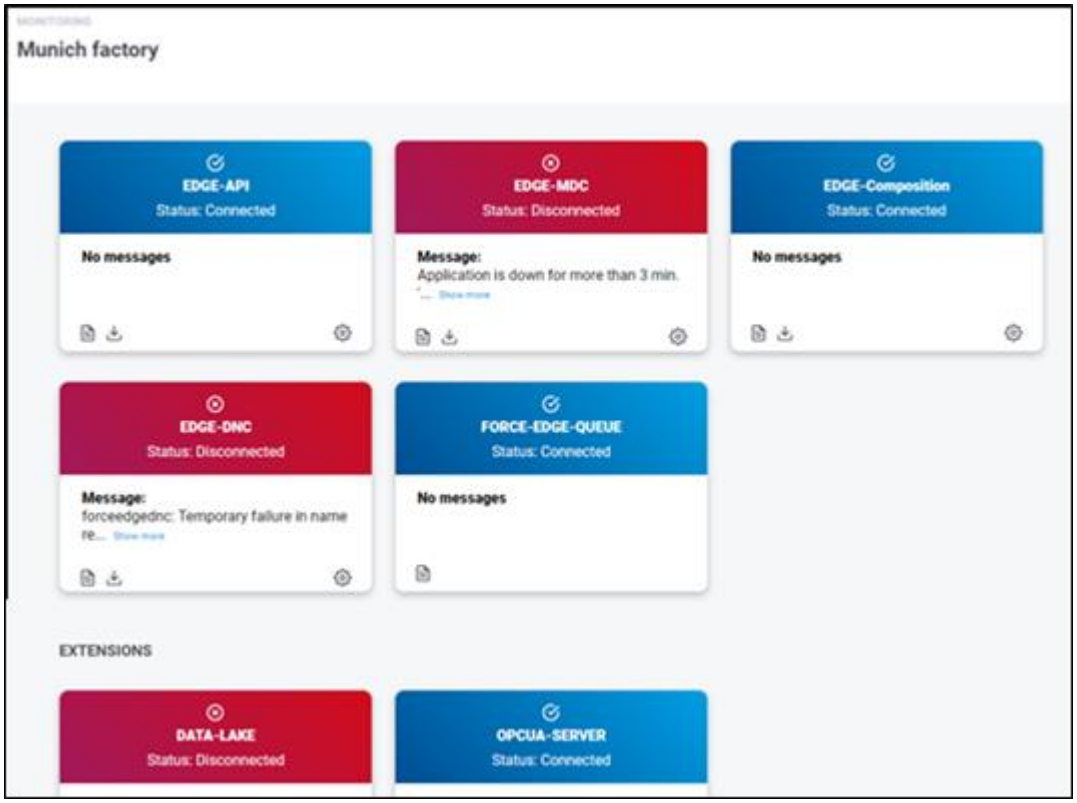


Fig. 34: Monitoring in EDGE CONNECT

Error messages and logs can be retrieved specifically for each component.

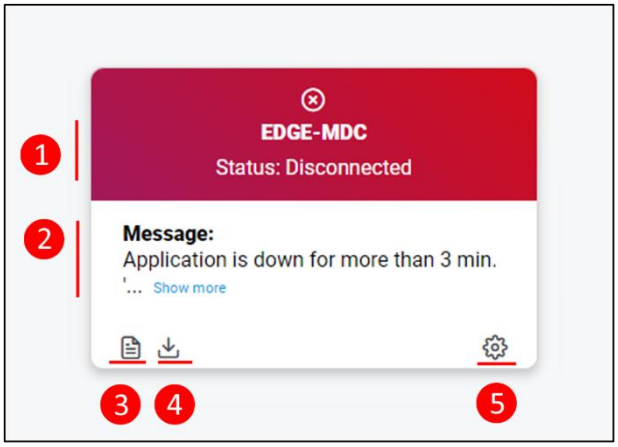


Figure 34: Component "EDGE-DNC" in the monitoring page

(1) Current status of the component
(2) Error indication via message:
Clicking Show **more** displays the full error message in a pop-up window.
(3) Displays the last warning and error message of the component for each case
(4) Enables downloading the log file for a specific day
(5) Setting the log-level

Log level

Log level	Description
Trace	All information from the system is stored in the log file. ⚠ Generates very large amounts of data.
Debug	Log file stores additional information from the system in addition to regular information (see Log level: Info)
Info	Log file contains information about the status of the asset, connection information, etc.
Warning	Only warnings are stored in the log file
Error	Only error messages are stored in the log file

8 Annex

8.1 Document conventions

Conventions	Description
Bold type	Buttons and options names are written in bold type.
Italics	Highlighted words are in italics.
Icons	For a function that is represented by an icon, the icon is referenced as the object.
Action result	Action results are indicated by ➔.
Prerequisites	Prerequisites are indicated by ✓.
Warnings	Warnings are indicated by ⚠.
Notes	Notes are indicated by ⓘ.
Tips	Tips are indicated by ⓣ.

Table 4: Fonts, formatting and characters used

8.2 Abbreviations and terms used

Abbreviation	Description
Apache Kafka	Apache Kafka is a distributed messaging system that uses the publish-subscribe method.
Asset	Generic term for all objects that EDGE CONNECT can connect (e.g., machines, sensors, databases or IT systems).
Brownfield	An existing factory or manufacturing facility that was already been built and has been in operation for some time. The Brownfield approach in the context of Industry 4.0 means the digital transformation of an existing manufacturing plant.
CP	Communication Processor
DB	Database
DNC	Distributed Numerical Control: NC systems that are connected to a computer. The individual systems can be centrally supplied with NC programs and then coordinated.
IT	Information technology
Machine	In EDGE CONNECT, a machine is a plant unit according to ISA 95 standard. If there are no further plant units (i.e., not additional physical controllers), it is referred to as a plant.
MDC	Machine Data Connection

Abbreviation	Description
MQTT	Message Queuing Telemetry Transport: Open network protocol for machine-to-machine (M2M) communication that enables transmitting telemetry data in the form of messages between devices, despite high delays or network limitations.
MR	Machine Repository
NATS.io	A connectivity system that uses the publish-subscribe method. There are different options to implement the messaging.
Northbound	A northbound interface communicates with a higher-level element in a particular network component.
OPC UA	Open Platform Communications Unified Architecture: platform independent service-oriented architecture that constitutes a standard for exchanging data.
OT	Operative technology - refers to hardware and software that monitors and controls the performance of physical devices.
Plug-in	FORCAM uses plug-ins as simplified connections to controllers.
POST	POST is a method which is supported by HTTP and means that a web server accepts the data contained in the body of the message requested.
PUT	The PUT method is used to update a resource available on the server. Typically, it replaces anything that exists at the target URL with something else.
REST	Representational State Transfer: Programming paradigm for distributed systems (collection of independent computers that present themselves to the user as a single system)
RESTful API	API for data exchange based on HTTP requests via GET, PUT, POST and DELETE, which is subject to the requirements or restrictions of the REST architecture.
Signal	Values read from the machine controller, such as temperature, pressure or certain statuses.
Southbound	Acting as the equivalent to the Northbound interface, a Southbound interface communicates with lower-level components.
SPS	Programmable Logical Control
TLS	Transport Layer Security Encryption protocol for the transport layer of the Internet. The data streams between client and server are encrypted. TLS is the successor protocol to SSL.
UDP	User Datagram Protocol - minimal, connectionless network protocolthat belongs to the transport layer of the Internet protocol family . UDP enables applications to send datagrams in IP-based computer networks.
UTC	Coordinated Universal Time
Wildcard	Placeholder for other characters

Table 5: Abbreviations and terms used

8.3 List of supported plug-ins

MDC Plug-ins

Name	Read	Write	Transmission type Polling/Event-based
AUDI SPS	X	X	X/
Controller for FORCAM FORCE DB tables	X		X/
CSV File Exchange	X		X/
Euromap 63	X		X/
Euromap 77 (via OPC UA)	X	X	/X
FANUC	X	X	X/
FORCAM I/O Controller	X	X	/X
FORCAM I/O Controller (hardware)	X		
MAKINO Pro 3/Pro 6	X		X/
MAZAK Mazatrol Fusion M640M	X	X	/X
MAZAK Mazatol Fusion M640MTPro	X	X	/X
MAZAK Mazatol Matrix	X	X	/X
MAZAK Mazatol Smart	X	X	/X
MAZAK Mazatol Smooth	X	X	/X
MCIS RPC (SINUMERIK 810D/840D/840D)	X		X/X
Modbus	X		
MQTT	X	X*	/X
MT Connect	X		X/
Node-RED	X	X	/X
OKUMA	X		X/
OMRON CS/CJ	X	X	X/
OMRON CV	X	X	X/

Name	Read	Write	Transmission type Polling/Event-based
OPC Classic	X	X	X/
OPC XML	X		X/
OPC UA	X	X	/X
Rockwell / Allen Bradley	X	X	X/
RPC	X	X	/X
Schneider Electric	X		X/
Siemens LOGO	X		X/
Siemens S5	X		X/
Siemens S7 (200, 300, 400, 1200, 1500)	X	X	X/
SQL Database Exchange	X		X/
Weihenstephan	X		X/
Wiesemann & Theis (WUT)	X		X/
Windows RPC	X	X	/X

Table 6: List of all supported machine connection variants

DNC Plug-ins

Name	Read	Write
COM	X	X
External program file transfer (Preview version)	X	X
FANUC	X	X
File Handler (File Copy)	X	X
File Handler Server	X	X
FTP Plug-in	X	X
Heidenhain	X	X
Mazak	X	X
MOXA Box	X	X
RPC (Preview version)	X	X
WUT (Preview version)	X	X

Table 7: List of all supported NC machine connection variants

8.4 Standardized events

Event type	Values	Function
General Information	– Machine ID – Machine Name – Exernal Machine ID – Reference (any) – Timestamp: – Type (any) – Value (any)	Any information
Impulse	– Machine ID – Machine name – Exernal machine ID – Reference (any) – Timestamp: – Count	E.g., stroke, shot
Quantity	– Machine ID – Machine name – Exernal machine ID – Reference (any) – Timestamp: – Amount – Unit (optional) – QualityDetail (optional)	Produced quantity
Signal package	– Machine ID – Machine name – Exernal machine ID – Reference (any) – Timestamp: – ARRAY [SignalName, Value, TimeStampUTC, Unit (optional)]	Collection of signals (e.g., serial number, pressure and temperature)
Signal value	– Machine ID – Machine name – Exernal machine ID – Reference (any) – Timestamp: – SignalName: – Value – Unit (optional)	Temperature, pressure, etc.
State	– Machine ID – Machine name – Exernal machine ID	State (production or downtime)

Event type	Values	Function
	– Reference (any) – Timestamp: – State (Production or Downtime) – StatusCodes (optional list of statuses)	

Table 8: Events and their function in EDGE CONNECT

8.5 Script examples

8.5.1 Asset status and temperature

The following script sends the status of the asset (production or stoppage). In addition, the temperature is also indicated. As soon as the temperature changes, the updated temperature is sent.

```
var_local

begin
        oldState: boolean;
        oldTemperature: string;
end;

oncepersecond
begin

        if( oldState!= @|PLC|@:DONE) then
        begin
                oldState := @|PLC|@:DONE;
                if @|PLC|@:DONE then
                begin
                        sendStateProduction()
                end
                else
                begin
                        sendStateStoppage();
                end;
        end;

        if( oldTemperature != toString(@|PLC|@:TEMP)) then
        begin
                oldTemperature := toString(@|PLC|@:TEMP);
                sendSignalValue("TEMPERATURE", toString(@|PLC|@:TEMP), "Degrees");
        end;

end;
```

8.5.2 Temperature and humidity

The following script sends the current temperature and humidity. This occurs in intervals of 30 seconds, and as soon as a change of these values takes place.

```
var_local
begin
        oldTemperature : string;
        oldHumidity : string;
        seconds: number;
end;

oncepersecond
begin
```

```
        seconds := seconds + 1;

        if (seconds > 30) then
        begin
                seconds := 0;
                oldTemperature := "";
                oldHumidity := "";
        end;

        if (oldTemperature != @|PLC|@:TEMP ) then
        begin
                oldTemperature := @|PLC|@:TEMP;
                sendSignalValue("TEMP", toString(@|PLC|@:TEMP), "Degree");
        end;

        if (oldHumidity != @|PLC|@:HUMIDITY ) then
        begin
                oldHumidity := @|PLC|@:HUMIDITY;
                sendSignalValue("HUMIDITY", toString(@|PLC|@:HUMIDITY), "Degree");
        end;
end;
```

8.5.3 Crane control

This script collects data from a crane control with a black, a green and a red button.

- The black button turns the machine on and off.
- The red button triggers an emergency.
- The green button sends a pulse for piece counts and then counts this number up.

```
var_local
begin
        // GENERAL LOGIC VARIABLES
        seconds:  number;
        // MACHINE STATE
        state: number;
        stateOld: number;
        // MACHINE STATUS REASON
        status_reason: string;
        status_reasonOld: string;
        // PIECE COUNT VARIABLES
        counter: number;
        counterOld: number;
        counterSend: number;
end;

begin
        //DEFINE LISTS START
        ListNew("STATUSCODES", "S");
        //DEFINE LISTS END
end;

begin
        // INITIALIZE SCRIPT VARIABLES START
        if not initialized and not offline(@|PLC|@) then
        begin
                status_reason := " ";
                status_reasonOld := " ";
                counter := @|PLC|@:Good_count;
                counterOld:= counter;
                ListClear("STATUSCODES");
                // Set initialized to perform initializing once
                initialized := true;
        end
        else if initialized then
        begin
                counter := @|PLC|@:Good_count;
                ListClear("STATUSCODES");
        end;
        // INITIALIZE SCRIPT VARIABLES END
```

```
// ACTIONS ONCE PER SECOND START
oncePerSecond
begin
        seconds := seconds + 1;
end;
// ACTIONS ONCE PER SECOND END

// DEFINITION STATE / STATUS_REASON START
if offline(@|PLC|@) then
begin
        state := "1";
        status_reason :='NOT_CONNECTED';
        seconds := 0;
end
else if not @|PLC|@:Emergency_ON then
begin
        state := "1";
        status_reason :='EMERGENCY_ON';
        seconds := 0;
end
else if not @|PLC|@:Machine_ON then
begin
        state := "2";
        status_reason := 'PRODUCTION'
        seconds := 0;
end
else
begin
        if seconds > karenzZeit then
        begin
                state := "1";
                status_reason := 'UNDEFINED_STOPPAGE'
                seconds := 0;
        end
end;
// DEFINITION state END

// DEFINITION COUNTER START
if counter >= counterOld then                   // Part counter on PLC is incremented
begin
        counterSend := counter - counterOld;
        counterOld := counter;
end
else if counter < counterOld then               // Part counter on PLC changes to negative
begin
        counterSend := 32768 - counterOld;
        counterOld := counter;
end;
// DEFINITION COUNTER END

// SEND state status_reason START
if state <> stateOld or status_reason <> status_reasonOld then
begin
        if state == 2 then
        begin
                ListAdd("STATUSCODES", status_reason);
                sendStateProduction("STATUSCODES");
        end
        else
        begin
                if == 1 then
                begin
                        ListAdd("STATUSCODES", status_reason);
                        sendStateStoppage("STATUSCODES");
                end
        end;
        debugOut("@|PLC|@" + "Send state: " + state);
        stateOld := state;
        status_reasonOld := status_reason;
end;
// SEND state status_reason END
```

```
        // SEND STROKES / QUANTITY START
        if counterSend > 0 and packetNo <> packetNoOld then
        begin
                debugOut("@|PLC|@" + "Send quantity: " + toString(counterSend));
                SendQuantity(counterSend);
                counterSend := 0;
        end;
        // SEND STROKES / QUANTITY END

        // LOGGING SIGNALS WHEN CHANGED START
        logstring := "@|PLC|@ Signals: " + " offline: "             + toString(offline(@|PLC|@))
                                        + " State: "                + toString(state)
                                        + " Status Reason: "        + toString(status_reason)
                                        + " Machine_ON: "           + tostring(@|PLC|@:Machine_ON)
                                        + " Emergency_ON: "         + tostring(@|PLC|@:Emergency_ON )
                                        + " COUNTER: "              + tostring(@|PLC|@:Good_count)
                                        + " seconds: "              + toString(seconds);

        if logString <> logstringOld then
        begin
                debugOut(logString);
                logstringOld := logString;
        end;
        // LOGGING SIGNALS WHEN CHANGED END
end;
```

8.5.4 Signal package

The following script is an example of signal packages:

```
//
// Task: Send machine state / status_reason / quantities to runtime
// Created: 2021-05-12
// Version: 1.0
// Author:  FORCAM MDC
//
// ---------------------------------------------------------------
//
// Incoming signals
// Reg1 = holding register  1
//
//
// Outgoing information
// //state         = machine state
// //STATUSCODES   = Status reason
// Reg1SEND   = just display holding register
////-------------------------------------------------------------

// VARIABLES
var_local
begin
// GENERAL  VARIABLES
    seconds: number;
    logstring: string;
    logstringOld: string;
//  SIGNAL VARIABLES
    H1Old: number;
    H2Old: number;
    H3Old: number;
    H4Old: number;
    H5Old: number;
    H6Old: number;
    H7Old: number;
    H8Old: number;
    H9Old: number;
    H10Old: number;
 // SCRIPT INIZIALIZING VARIABLES
    initialized: boolean;
end;
```

```
begin
     if not initialized and not offline(@|PLC|@) then
    begin
    //DEFINE LISTS START (S=strin B=boolean N=number)
        ListNew("Signals", "S");
        ListNew("Values", "S");
//        ListNew("Timestamps", "S");
    //DEFINE LISTS END
    end;

// INITIALIZE SCRIPT & VARIABLES START
    if not initialized and not offline(@|PLC|@) then
    begin
          H1Old := 0;
          H2Old := 0;
          H3Old := 0;
          H4Old := 0;
          H5Old := 0;
          H6Old := 0;
          H7Old := 0;
          H8Old := 0;
          H9Old := 0;
          H10Old := 0;
          ListClear("Signals");
          ListClear("Values");
 //         ListClear("Timestamps");
//    set initialized to perform initializing once
      initialized := true;
    end
    else if initialized then


// ACTIONS ONCE PER SECOND START
   oncePerSecond
   begin
     seconds:= seconds + 1;

// ACTIONS ONCE PER SECOND END
// send one package for all 10 holding registers  for now always
// Reg1Content Start
    if( H1Old <> @|PLC|@:H1 ) then
   begin

 //fill lists
      H1Old := @|PLC|@:H1;
       ListAdd("Signals", "H1");
      ListAdd("Values", toString(@|PLC|@:H1));
      H2Old := @|PLC|@:H2;
      ListAdd("Signals", "H2");
      ListAdd("Values", toString(@|PLC|@:H2));
      H3Old := @|PLC|@:H3;
      ListAdd("Signals", "H3");
      ListAdd("Values", toString(@|PLC|@:H3));
      H4Old := @|PLC|@:H4;
      ListAdd("Signals", "H4");
      ListAdd("Values", toString(@|PLC|@:H4));
 //     H5Old := @|PLC|@:H5;
 //     ListAdd("Signals", "H5");
 //     ListAdd("Values", toString(@|PLC|@:H5));
 //     H6Old := @|PLC|@:Reg6;
 //     ListAdd("Signals", "H6");
 //     ListAdd("Values", toString(@|PLC|@:H6));
 //     H7Old := @|PLC|@:H7;
 //     ListAdd("Signals", "H7");
 //     ListAdd("Values", toString(@|PLC|@:H7));
 //     H8Old := @|PLC|@:H8;
 //     ListAdd("Signals", "H8");
 //     ListAdd("Values", toString(@|PLC|@:Reg8));
 //     H9Old := @|PLC|@:H9;
 //     ListAdd("Signals", "H9");
 //     ListAdd("Values", toString(@|PLC|@:Reg9));
 //   H10Old := @|PLC|@:H10;
 //    ListAdd("Signals", "H10");
 //     ListAdd("Values", toString(@|PLC|@:H10));
```

```
//      sendSignalValue("HoldingReg1", toString(@|PLC|@:H1));
 //     sendSignalValue("HoldingReg2", toString(@|PLC|@:H2));
 //     sendSignalValue("HoldingReg3", toString(@|PLC|@:H3));
//      sendSignalValue("HoldingReg4", toString(@|PLC|@:H4));
 //     sendSignalValue("HoldingReg10", toString(@|PLC|@:H10));
//  send Signal Package with lists
                                          SendSignalPackage("Signals", "Values")
       //initialize list
      begin
         ListClear("Signals");
         ListClear("Values");
      end;

// SENDING Holding Register  END

// LOGGING SIGNALS WHEN CHANGED START
    logstring := "@|PLC|@ Signals: "
                                      + " Reg 1: "              + tostring(@|PLC|@:H1)
                                      + " Reg 2: "              + tostring(@|PLC|@:H2)
                                      + " Reg 3: "              + tostring(@|PLC|@:H3)
                                      + " Reg 4: "              + tostring(@|PLC|@:H4)
   //                                   + " Reg 5: "              + tostring(@|PLC|@:H5)
   //                                   + " Reg 6: "              + tostring(@|PLC|@:H6)
   //                                   + " Reg 7: "              + tostring(@|PLC|@:H7)
   //                                   + " Reg 8: "              + tostring(@|PLC|@:H8)
   //                                   + " Reg 9: "              + tostring(@|PLC|@:H9)
   //                                   + " Reg 10: "             + tostring(@|PLC|@:H10)
                                                                       ;
    if logString <> logstringOld then
    begin
      debugOut(logString);
      logstringOld := logString;
    end;
// LOGGING SIGNALS WHEN CHANGED END
end;
end;
end;
```

8.6 Script functions

Usage	Script function Parameters in [...] are optional	Description	Output event
Default	SendImpulse(ImpulseCount, [Reference])	Sends impulses.	Impulses.
Default	SendQuantity(Quantity, [Unit], [QualityDetail], [Reference])	Sends a quantity.	Quantity
Custom	SendState(State, [StatusCodesListName], [Reference])	Sends a status.	State
Default	SendStateProduction([StatusCodesListName], [Reference])	Sends the productions status.	State
Default	SendStateStoppage([StatusCodesListName], [Reference])	Sends the stop state.	State
Default	SendSignalValue(SignalName, Value, [Unit], [Reference], [CustomerSpecificSetting], [Timestamp])	Sends the value of a signal. Data type "Long" (L) must be used for the timestamp list.	SignalValue
Default	SendSignalPackage(SignalNamesListName, ValuesListName, [UnitsListName], [Reference], [CustomerSpecificSetting], [TimestampsListName])	Sends signal values as a package. Data type "Long" (L) must be used for the timestamp list.	SignalPackage
Custom	SendGenericInformation(ParamName, ParamValue, [Reference])	Sends generic information.	GenericInformation
Helper	ListNew(ListName, DataType)	Creates a new list with the name ListName and list elements of the data type DataType (S for string, B for boolean, N for number).	-
Helper	ListAdd(ListName, Value)	Adds an element to the list.	-
Helper	ListClear(ListName)	Empties the list.	-
Helper	ListDelete(ListName)	Deletes the list.	-

Usage	Script function Parameters in [...] are optional	Description	Output event
Helper	GetMachineStatus()	Indicates the asset status.	-
Helper	GetMachineData(ParameterName)	Indicates asset data for the specified parameter.	-
Helper	SetParameter(ParameterName, ParameterValue)	Sets a new value for the specified parameter.	-
Helper	GetParameter(ParameterName)	Fetches the value for the specified parameter.	-
Helper	DeleteParameter(ParameterName)	Deletes the parameter.	-
Helper	DeleteAllParameters()	Deletes all parameters.	-
Helper	OFFLINE	Indicator whether the controller is offline or not.	-
Helper	IPADDRESS	The IP address of the Composition.	-
Helper	HOSTNAME	Host name of the Composition.	-
Helper	SQRT(args)	Root function MATH.	-
Helper	SIN(args)	Sine function MATH.	-
Helper	COS(args)	Cosine function MATH.	-
Helper	TAN(args)	Tangent function MATH.	-
Helper	RISINGEDGE(args)	At the beginning the variable is FALSE, the EDGE checks if the values have changed. If this is the case, the variable is corrected to TRUE.	-

Usage	Script function Parameters in [...] are optional	Description	Output event
Helper	FALLINGEDGE(args)	At the beginning the variable is TRUE, the EDGE checks if the values have changed. If this is the case, the variable is corrected to FALSE.	-
Helper	SUBSTRING(str, startIndex[, endIndex])	Substring of the specified string.	-
Helper	TONUMBER(str)	String to number (double), replaces comma to period in string.	-
Helper	TOSTRING(str or number[, formatSpecifier])	Specifies the format of the form width. The default formatting is used for empty strings. Width is the minimum length of the result string. Precision is the number of decimal places. If not specified, 0 is used. If the format specification starts with 0, the result string is prefixed with filled zeros. If the format specification ends with X, the number is converted to hexadecimal, using upper or lower case letters with upper or lower case x. In this case, the decimal places are always cut off.	-
Helper	LENGTH(obj)	The length of an object as a string value.	-
Helper	FORMATTIME(timeformatStr, timeOffset, [, timeunit])	Formats the current time with the time unit as one of the following: MILLISECOND SECOND MINUTE HOUR DAY MONTH YEAR MSABSOLUTE (current time) "R" at Format is specified as a number in milliseconds, otherwise the format is used and the offset and time unit are used to calculate the time.	-

Usage	Script function Parameters in [...] are optional	Description	Output event
Helper	STDLOG(ignored, logLevel, suffixNumber, logText)	The first parameter is ignored. The log level should be W = warning, C or F = error and everything else for the debug level. The suffix number, if not 0, is added to the end of the log text as "(<SuffixNumber>)" with script loggers.	-
Helper	DEBUGOUT(text)	Logs the text at debug log level with parser logger.	-
Helper	COPYFILE(inFile, outFile)	Copies data from in-file to out-file. Arguments can be file paths. If successful, the last modified out-file is also updated as in-file.	-
Helper	COPYREPLACE(inFile, outFile, searchStr, replaceStr)	Copies from in-file to out-file as with function COPYFILE, replacing all incidences of search-string with replace-string.	-
Helper	ATTIME(seconds, obj)	Calculates the object every day at specified times in the time format (hours: minutes: seconds)	-
Helper	FROMASCII(num)	Returns a string that has the numeric value specified as num.	-
Helper	SLEEP(ms)	Pauses the current thread for a specified time in milliseconds (ms).	-