FORCE EDGE CONNECT

Version: 230406

Manual

Document: Manual- FORCE EDGE CONNECT

Release date: 2023-04-06

Document version: 1

Author: FORCAM GmbH

Contents

1 Concept

FORCE EDGE CONNECT (hereafter simply referred to as EDGE CONNECT) offers manufacturing companies a solution for digitally connecting their heterogeneous machinery. Almost all assets can be digitized with EDGE CONNECT, regardless of age or technical status. 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 a manufacturing plant in the 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 makes a significant contribution to the digital transformation by closing the gap between IT (information technology) and OT (operational technology).

EDGE CONNECT interconnects the various asset connections and signals and delivers them as standardized events to superordinate systems. These can be ME (Manufacturing Execution) or MOM (Manufacturing Operation Management) systems such as SAP DMC/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 via a plugin concept for easier future expansion. 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). The FORCAM I/O Controller is available as separate hardware for digitizing the asset if the asset is not network-compatible. FORCAM FORCE EDGE is continually expanded with plugins in order to meet the challenge of digitally mapping every type of asset via EDGE CONNECT.

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 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 EDGE Composition Layer. 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 component makes it easy to create and use templates. This means that templates can be defined for asset connections or derived from existing connections and used for the connection of the same machine types. This further reduces the individual effort required to connect an asset, enabling the time- and resource-efficient implementation of digitization projects. The template structure ensures a standardized connection of identical assets, thus enabling the comparison of assets of the same type.

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.

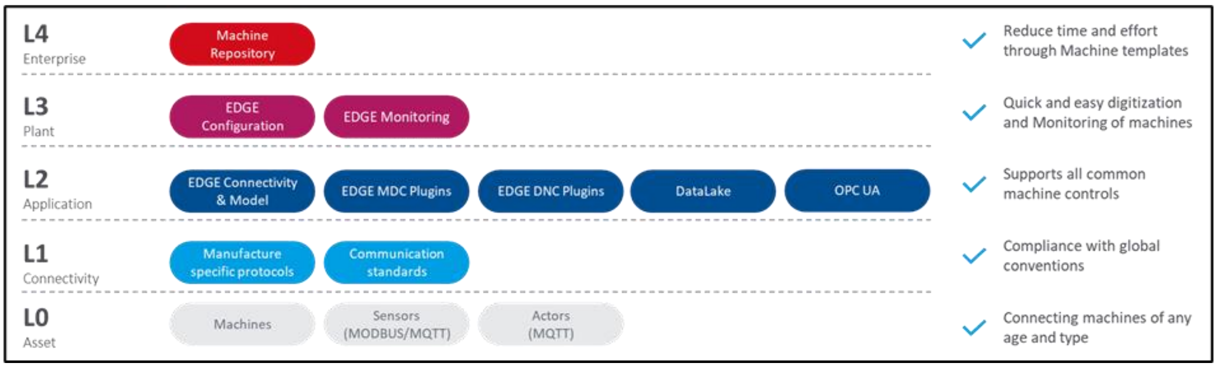

L4
Enterprise
Machine Repository
L3
Plant
EDGE Configuration
EDGE Monitoring
L2
Application
EDGE Connectivity & Model
EDGE MDC Plugins
EDGE DNC Plugins
DataLake
OPC UA
L1
Connectivity
Manufacture specific protocols
Communication standards
L0
Asset
Machines
Sensors (MODBUS/MQTT)
Actors (MQTT)
Reduce time and effort through Machine templates
Quick and easy digitization and Monitoring of machines
Supports all common machine controls
Compliance with global conventions
Connecting machines of any age and type

Fig. 1: Location of the EDGE CONNECT solution components

Figure 1 shows the reference architecture of the Open Industry 4.0 Alliance, which is also the basis of the EDGE CONNECT architecture. FORCAM contributes significantly to digitalization in industry and focuses on customer benefits. The connectivity of hardware through intuitive and user-friendly software is what makes EDGE CONNECT stand out.

2 System components

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

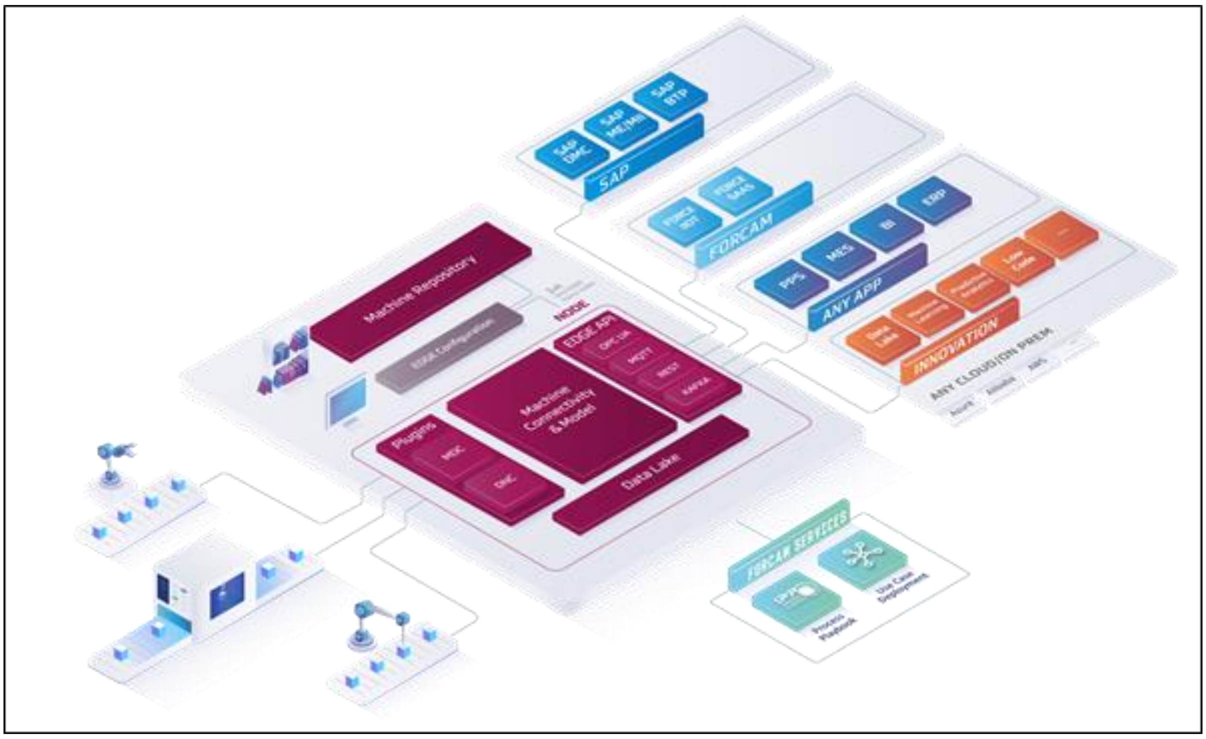


Fig. 2: Schematic structure of EDGE CONNECT

2.1 EDGE Node

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

2.1.1 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.
The following three components manage the communication:

Plugins

The plugins used in the EDGE CONNECT establish communication links with specific machine controllers. They also standardize the data, thus making evaluations more comparable.
They allow direct communication with various machine controllers, but also cover modern communication protocols such as MQTT, UPC UA and many more. The plugin concept of EDGE CONNECT is extensible, FORCAM is continuously expanding the number of supported plugins.

The plugins are divided into those for Machine Data Collection (MDC) and for Distributed Numerical Control (DNC).
MDC plugins include those designed for unidirectional readout of machine signals as well as for bidirectional signal transmission, i.e., for reading out and writing back signals.
DNC plugins are used for transferring and reading NC files. They are used for transferring NC programs to the machine's file system or to query the program active on the machine.

For the most common control types, a set of plugins is included in EDGE CONNECT by default. An overview of the current FORCAM plugins is listed in chapter 8.3.

EDGE MDC Layer

The EDGE MDC layer manages the actual connection of the machines. The essential elements are the selection of the suitable plugin for the communication with the asset control, the configuration of the machine master data, the setting of the network connection and the definition of the asset signals. In addition, the EDGE MDC Layer forwards asset signals to the EDGE Composition Layer.

EDGE DNC Layer

The EDGE DNC Layer manages the actual connection of machines with an NC supply. The essential elements here are the selection of the suitable plugin for communication with the asset control, the configuration of the asset master data, the setting of the network connection and the configuration of the DNC supply.

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

2.1.2 Signal Composition

The EDGE Signal Composition Layer is used to derive logical asset states. Either a script language or a graphical solution can be used to derive standardized events from signal combinations. For this purpose, the Composition Layer unifies the reporting capabilities. In addition, individual events are made possible. The Composition also makes it possible to react to events and to write values to the control unit of the asset, as far as 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 latter provides an easy introduction into the world of signal composition.

2.1.3 Northbound Link

The Northbound Link component is responsible for the communication between EDGE CONNECT and any 3rd 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.

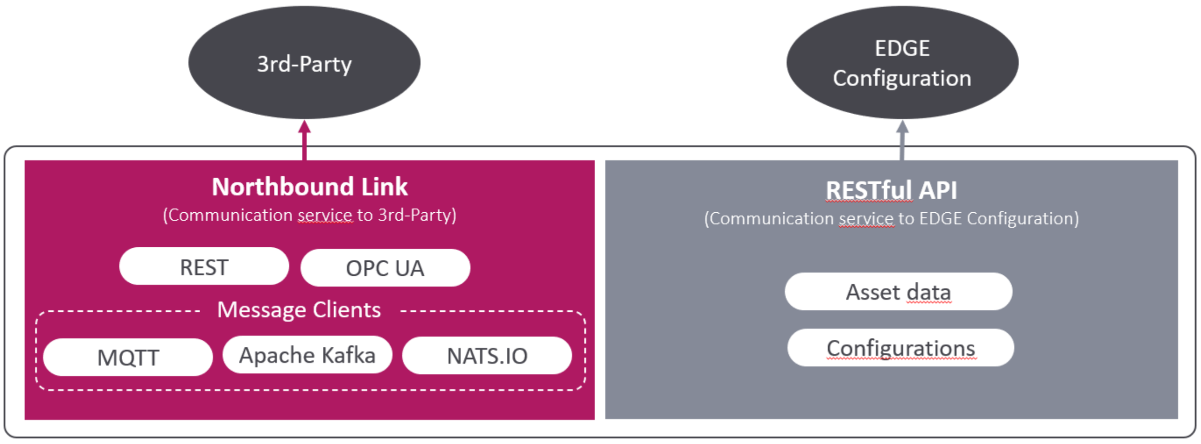


Fig. 3: Northbound Link

RESTful API

The RESTful API is used to retrieve asset master data and to configure the asset connections. It is mainly responsible for the communication between EDGE NODE and EDGE Configuration.

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

⚠ The middleware 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 the signal level, the interpretation level and the event level, including configuration changes, 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.

2.2 EDGE Configuration

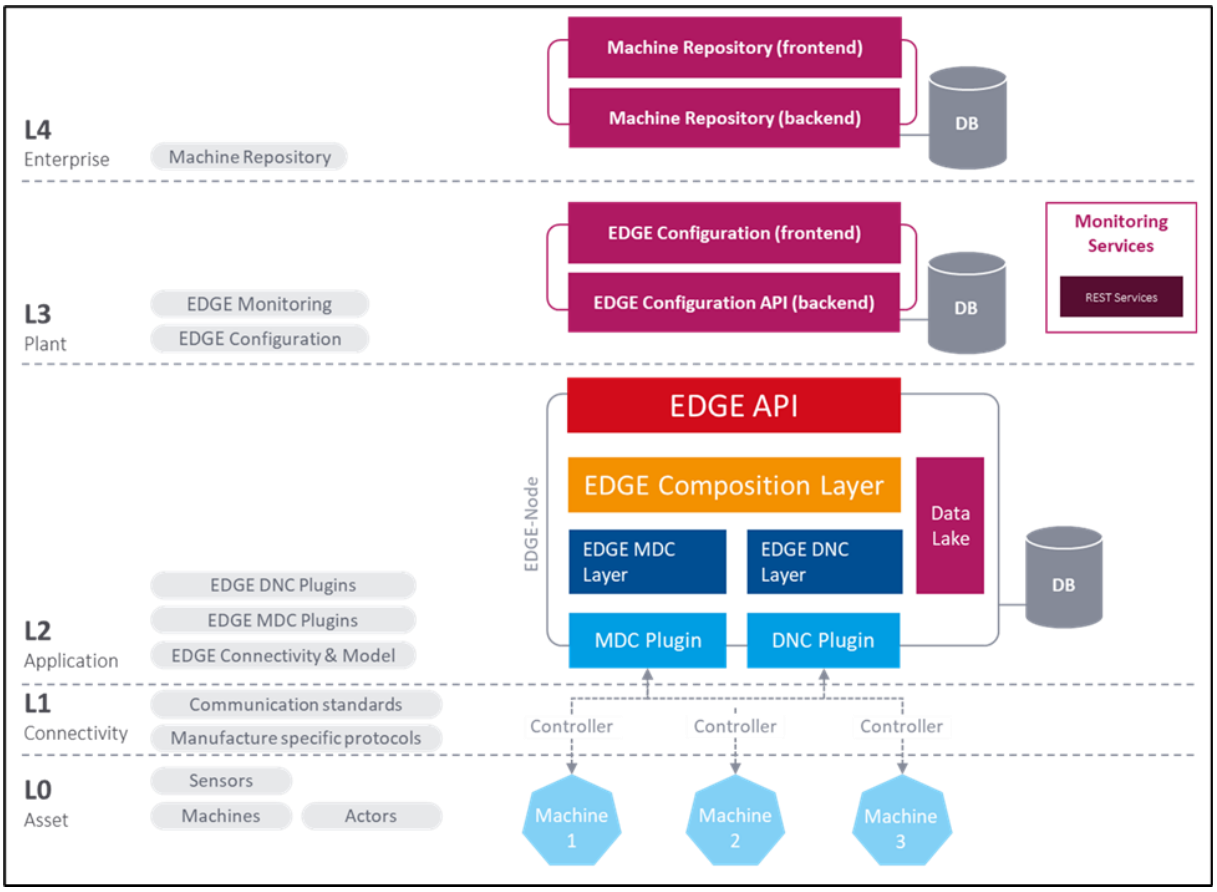
EDGE Configuration is the management interface for EDGE CONNECT. It can be used to manage multiple EDGE nodes. An EDGE node is the bundling of signal collection from several assets. Depending on the amount of data, one or more EDGE nodes are used per plant. The management of the nodes is done centrally.

2.3 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 machine-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.

3 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 plugins 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:

- **EDGE MDC Layer** controls the connection of the asset via **Plugins** u and forwards the signals to **EDGE Composition Layer**. Similarly, DNC-capable assets are connected by **EDGE DNC Layer**.
- **EDGE Composition Layer** is responsible for signal interpretation and for creating the standardized events.

- **EDGE API** is the programming interface through which assets, event management and notification of third-party systems can be configured "northbound".
- **Data Lake** records all data, configuration changes as well as write operations and transferred NC data. The data can be accessed via the Data Lake API.
- **DB** contains all configurations related to the asset connections.

Level 3 - Plant

The configuration component can serve 1 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. 4: Options for installing EDGE CONNECT

For the installation, an installer is provided containing the **EDGE Configuration**, **EDGE Node** and **Machine Repository**. 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 controllers 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 must be purchased in addition to EDGE CONNECT.

SAP BTP

EDGE CONNECT can be purchased as a solution extension of SAP Digital Manufacturing Cloud (SAP DMC) on the SAP Business Technology Platform (SAP BTP).
If purchased, FORCAM provides the hardware on which EDGE CONNECT is run. A Microsoft Azure Stack EDGE (ASE) is used for this purpose, which is preconfigured by FORCAM. To do so, FORCAM requires specific IP addresses, which must be entered manually by the customer or communicated to FORCAM's service providers. The customer is responsible for integrating the ASE into the existing hardware environment. FORCAM guarantees the function and availability of the ASE and the software components.

5 Basic settings

General settings for EDGE CONNECT are made in the **Home** section.
Besides introducing the handling of tables, this chapter provides information on the following aspects:

- User administration
- Supplied master data
- Licensing
- Download area
- Monitoring

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

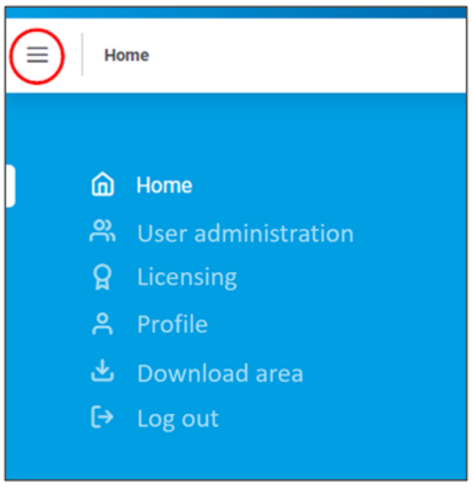
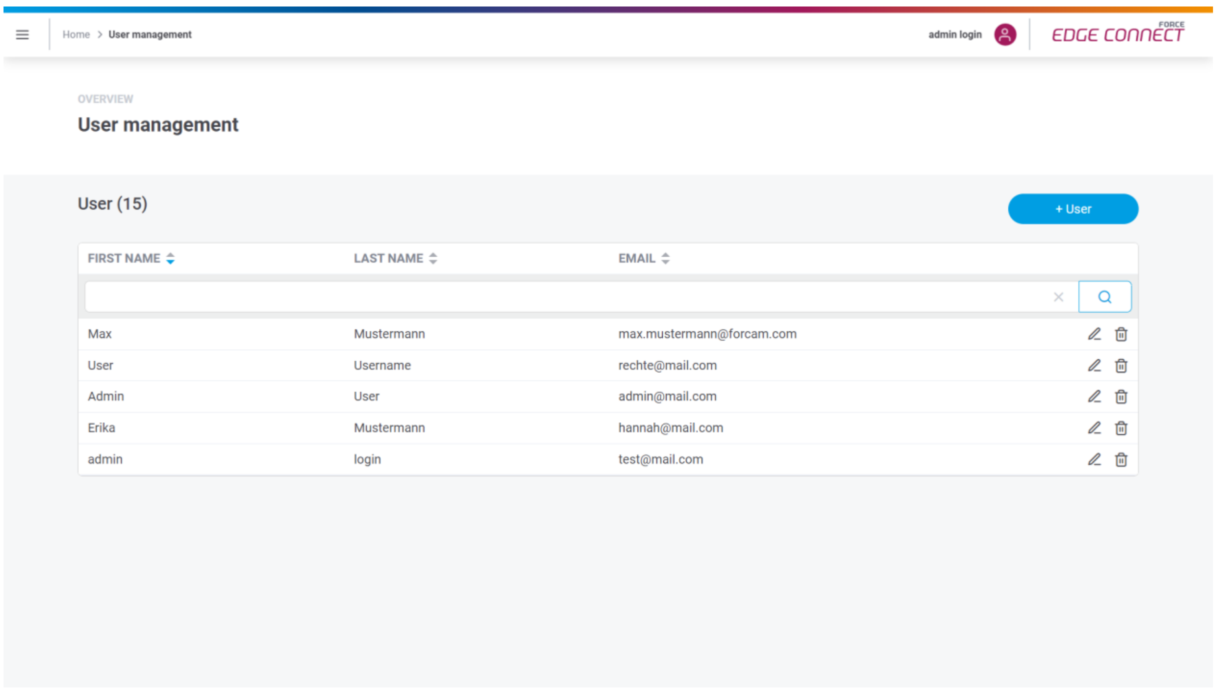


Fig. 5: Calling up the Home section

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.



User management

User (15)

FIRST NAME	LAST NAME	EMAIL
Max	Mustermann	max.mustermann@forcam.com
User	Username	rechte@mail.com
Admin	User	admin@mail.com
Erika	Mustermann	hannah@mail.com
admin	login	test@mail.com

Fig. 6: EDGE CONNECT user administration with 2 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 below).
5. Save.

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

Table 1: User permissions in EDGE CONNECT

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 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.

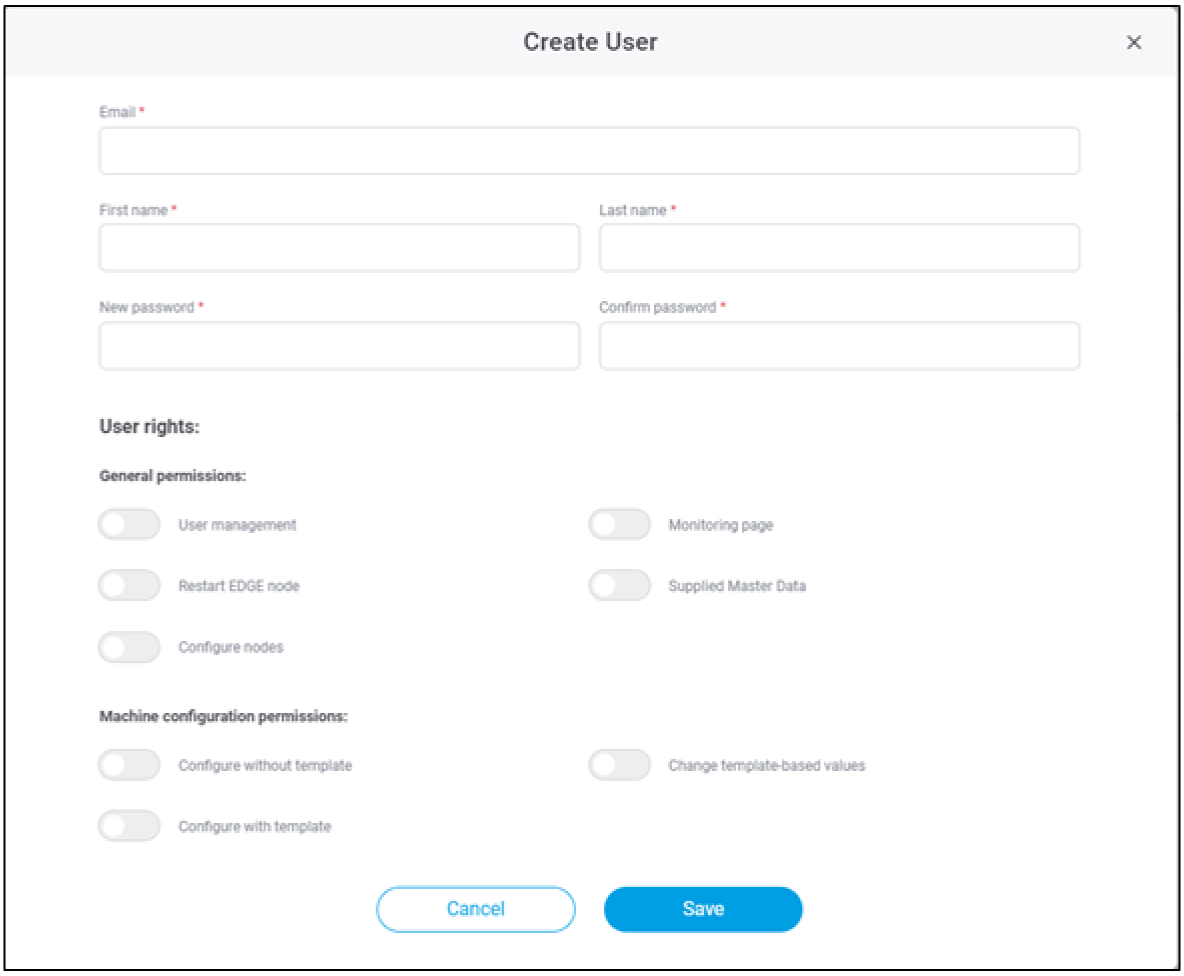


Fig. 7: Dialog for creating a new user

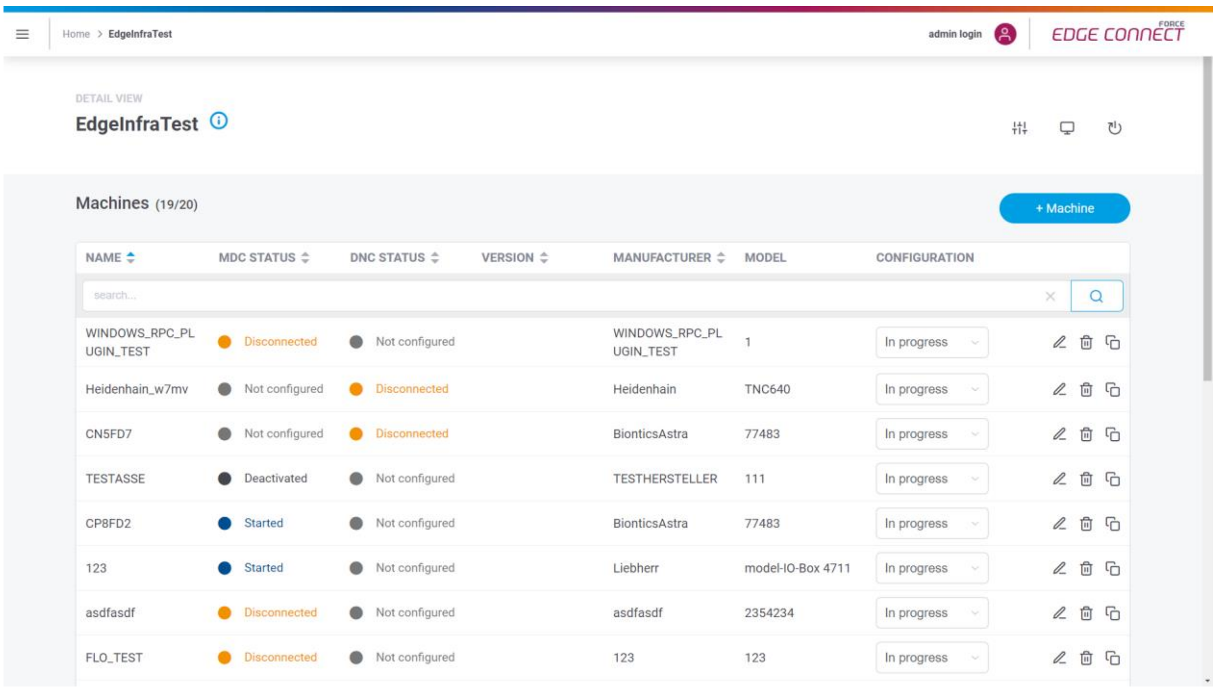
5.2 Supplied master data

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 DMC. 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.

When the master data is sent in the third-party application, EDGE Configuration uses the API to create an asset in EDGE CONNECT. All new assets are displayed in 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.



EdgeInfraTest

Machines (19/20)

NAME	MDC STATUS	DNC STATUS	VERSION	MANUFACTURER	MODEL	CONFIGURATION
WINDOWS_RPC_PLUGIN_TEST	Disconnected	Not configured		WINDOWS_RPC_PLUGIN_TEST	1	In progress
Heidenhain_w7mv	Not configured	Disconnected		Heidenhain	TNC640	In progress
CN5FD7	Not configured	Disconnected		BionticsAstra	77483	In progress
TESTASSE	Deactivated	Not configured		TESTHERSTELLER	111	In progress
CP8FD2	Started	Not configured		BionticsAstra	77483	In progress
123	Started	Not configured		Liebherr	model-IO-Box 4711	In progress
asdfasdf	Disconnected	Not configured		asdfasdf	2354234	In progress
FLO_TEST	Disconnected	Not configured		123	123	In progress

Fig. 8: List of assets with Supplied Master Data

The following master data can be created via the API:

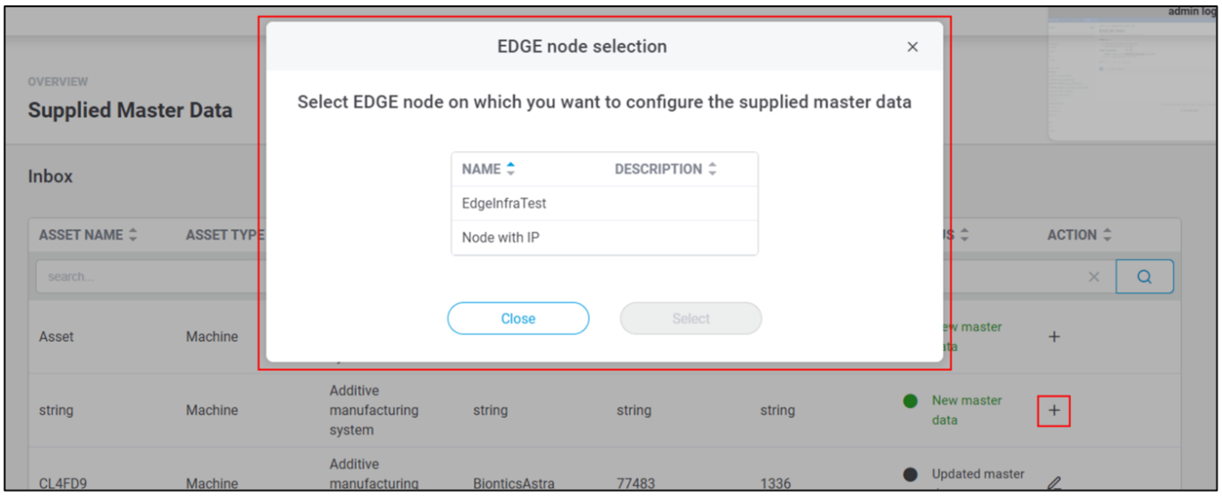
- Asset name
- Asset type
- Asset class
- Manufacturer
- Model
- Serial number

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: If a template is selected when configuring an asset created through the API, the master data passed through the API is used and that of the template is discarded.

To create new master data on a node:

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


OVERVIEW
Supplied Master Data
Inbox

EDGE node selection

Select EDGE node on which you want to configure the supplied master data

NAME	DESCRIPTION
EdgeInfraTest	
Node with IP	

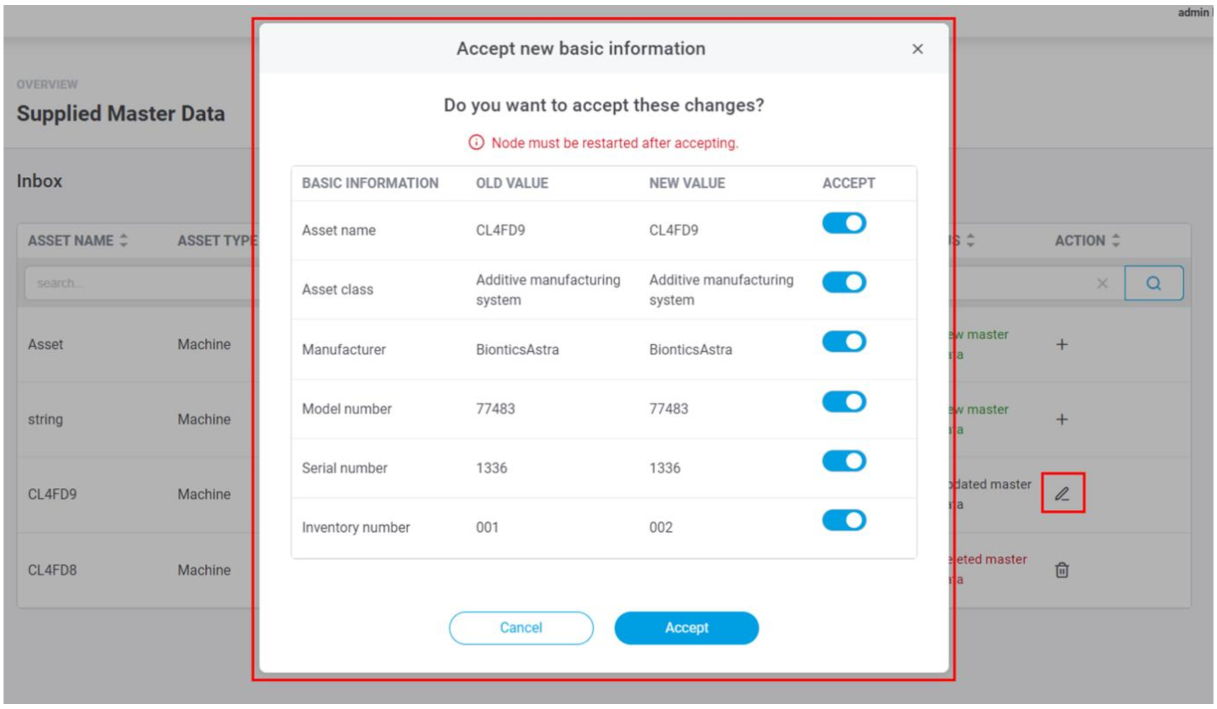
Fig. 9: Selection of an EDGE node for the creation of supplied master data

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 on the pencil icon to 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. Result
5. Restart the corresponding node.



Do you want to accept these changes?

Node must be restarted after accepting.

BASIC INFORMATION	OLD VALUE	NEW VALUE	ACCEPT
Asset name	CL4FD9	CL4FD9	✓
Asset class	Additive manufacturing system	Additive manufacturing system	✓
Manufacturer	BionticsAstra	BionticsAstra	✓
Model number	77483	77483	✓
Serial number	1336	1336	✓
Inventory number	001	002	✓

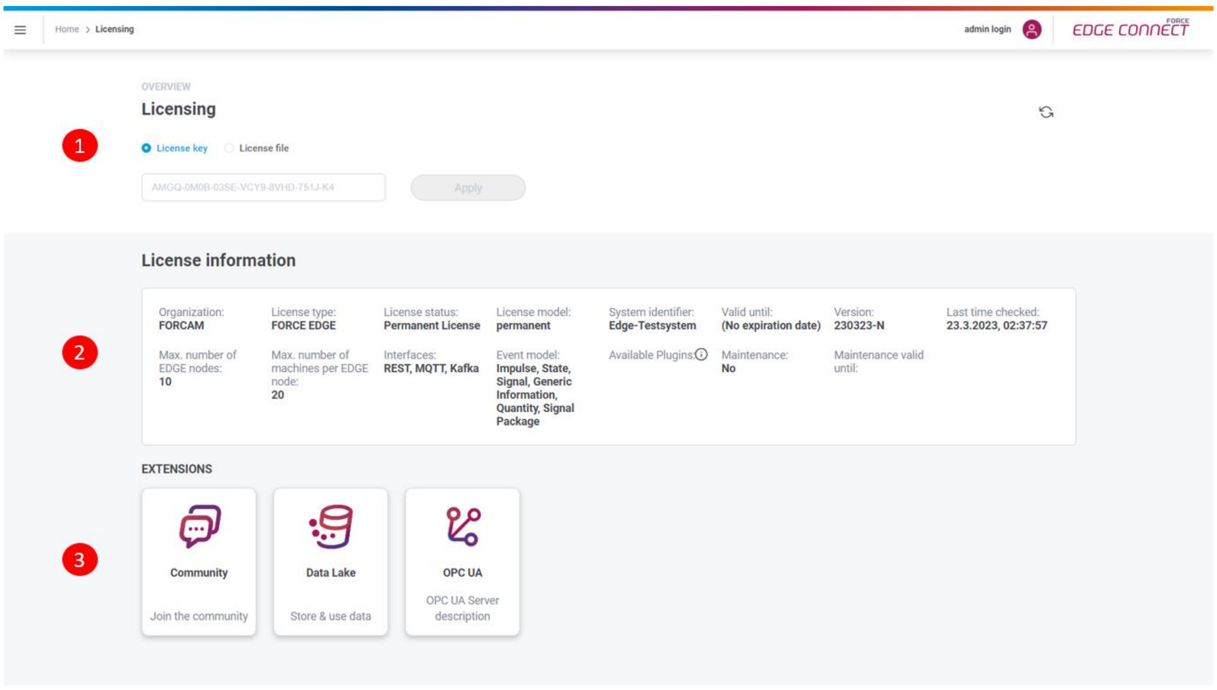
Fig. 10: Confirmation of asset master data changes

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**.



OVERVIEW

Licensing

License key License file

AMGQ-0M0B-03SE-VCY9-8VHD-751J-K4 Apply

License information

Organization: **FORCAM**	License type: **FORCE EDGE**	License status: **Permanent License**	License model: **permanent**	System identifier: **Edge-Testsystem**	Valid until: **(No expiration date)**	Version: **230323-N**	Last time checked: **23.3.2023, 02:37:57**
Max. number of EDGE nodes: **10**	Max. number of machines per EDGE node: **20**	Interfaces: **REST, MQTT, Kafka**	Event model: **Impulse, State, Signal, Generic Information, Quantity, Signal Package**	Available Plugins:	Maintenance: **No**	Maintenance valid until:	

EXTENSIONS

Community	Data Lake	OPC UA
Join the community	Store & use data	OPC UA Server description

Fig. 11: 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

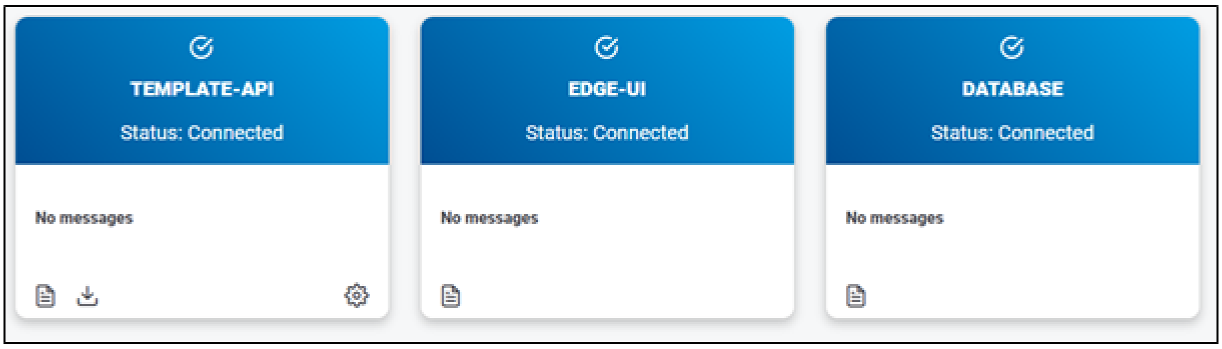
The current EDGE CONNECT documentation can be downloaded in several languages from the **General** tab. Currently, the user manual and a product description are available. The manual is the document at hand, with detailed configuration instructions. The product description is a shorter document describing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

In the **MDC Plugins** and **DNC Plugins** tabs, FORCAM provides additional applications. They are needed to communicate with an asset via the corresponding plugin. In terms of software architecture, the applications are located between the EDGE MDC Layer and the asset, enabling bidirectional communication.

5.5 Monitoring

The monitoring in the Home menu is used to monitor the EDGE Configuration. The monitoring of the EDGE 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.


TEMPLATE-API
Status: Connected
No messages

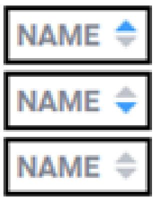
EDGE-UI
Status: Connected
No messages

DATABASE
Status: Connected
No messages

Fig. 12: Monitoring template transmissions via the API

5.6 Table Sorting

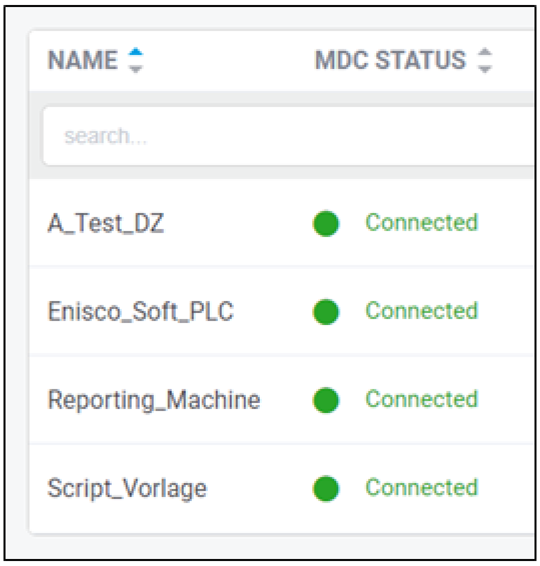
Most pages in EDGE CONNECT display data in the form of tables. To offer the familiar ease of use you know from other table tools, the sorting function of columns has been used here as well: You can sort the columns alphabetically in ascending or descending order.



NAME	Ascending
NAME	Descending
NAME	No sorting

Fig. 13: 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.



NAME	MDC STATUS
search...	
A_Test_DZ	● Connected
Enisco_Soft_PLC	● Connected
Reporting_Machine	● Connected
Script_Vorlage	● Connected

Fig. 14: Alphabetical sorting and content grouping

6 EDGE configurations

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.

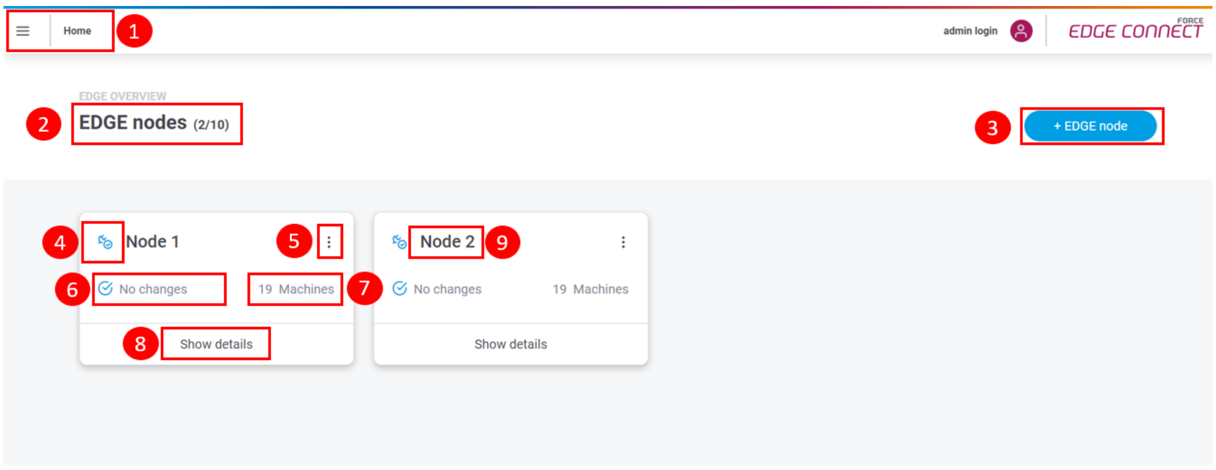


Fig. 15: EDGE CONNECT entry and overview page

(1) EDGE CONNECT Home menu
- User administration
- Supplied master data
- Licensing
- Profile
- Download area
- Monitoring

(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

ⓘ Changes to the user administration regarding user rights can take up to 30 minutes until they take effect throughout the system.

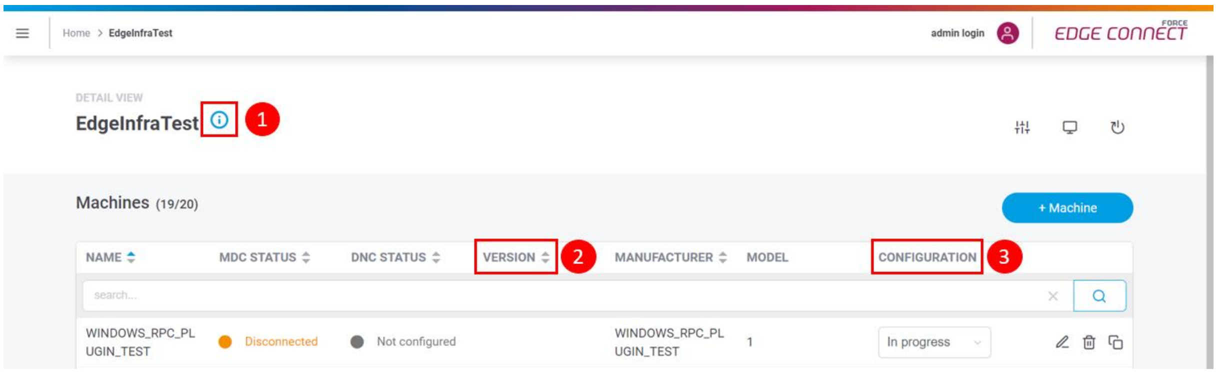


Fig.16: Asset overview as next page after clicking on "EDGE Node"

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

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

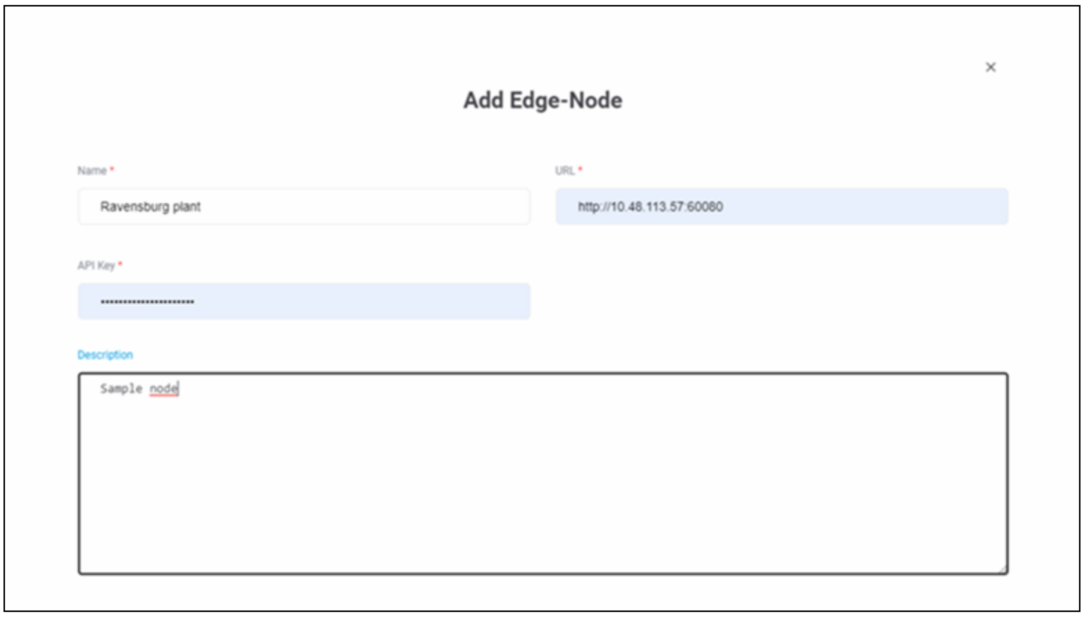
CONFIGURATION lets you manually determine which status the configuration should have, for example to give employees an overview or to call them in:

- In progress:
  The configuration is not yet complete and is to be continued at another time.
- In validation:
  The configuration of the asset is to 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 corresponds to an instance of a connection variant. There can be several nodes per plant. They are logically bundled so that the asset workload is 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.



Add Edge-Node

Name *
Ravensburg plant

URL *
http://10.48.113.57:60080

API Key *
•••••••••••••••••••••

Description
Sample node

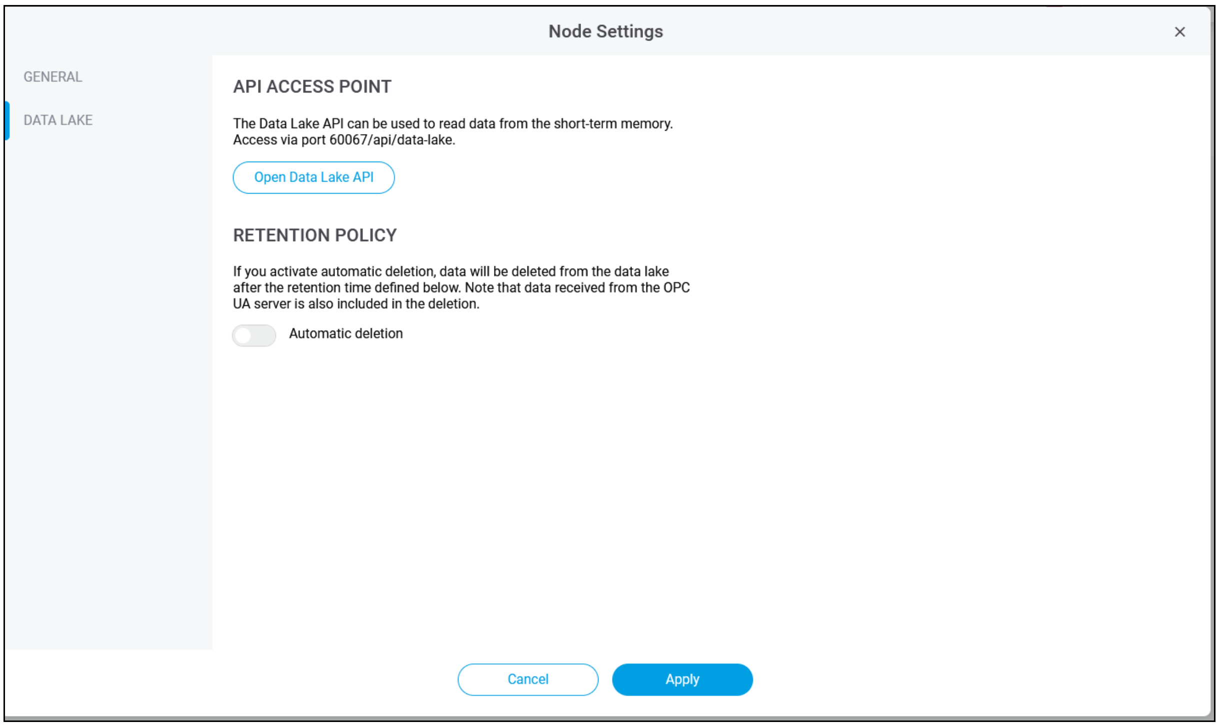
Fig. 17: Dialog for adding a new node

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.
   - 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.



The Data Lake API can be used to read data from the short-term memory.
Access via port 60067/api/data-lake.

If you activate automatic deletion, data will be deleted from the data lake after the retention time defined below. Note that data received from the OPC UA server is also included in the deletion.

Fig. 18: 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. Edit the values.
5. Save.

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. The 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.

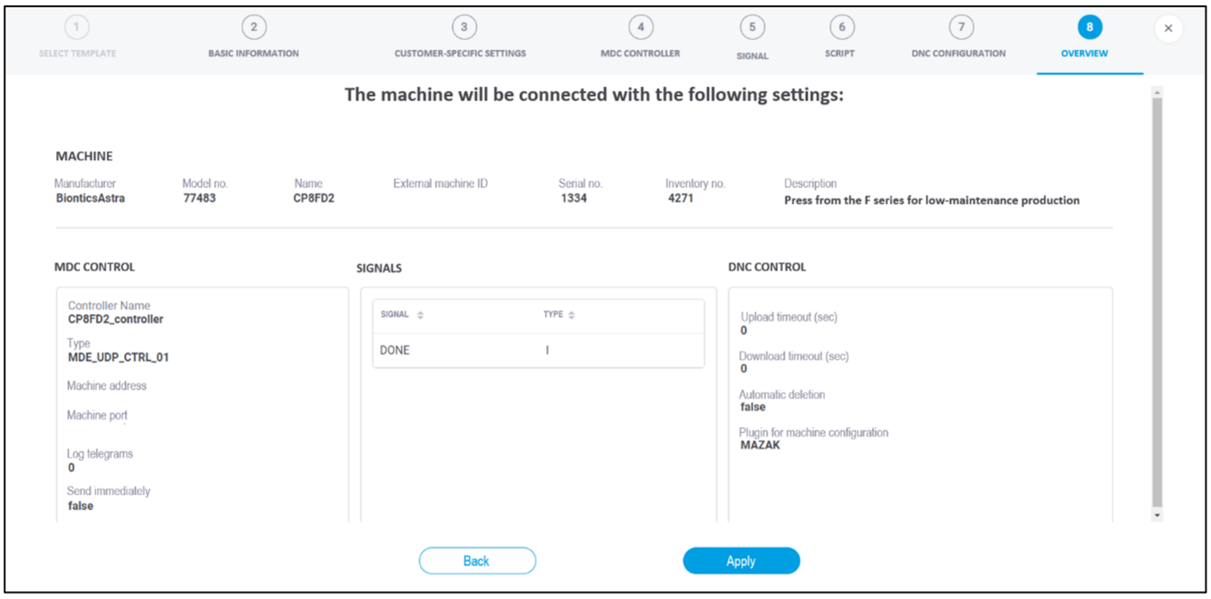
ⓘ The Data Lake should be considered a short-term memory, the related technical requirements stated in the System Requirements must be taken into account!

6.3 Add asset

The dialog for adding an asset guides you through eight steps necessary for a connection. 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.



The machine will be connected with the following settings:

MACHINE

Manufacturer	Model no.	Name	External machine ID	Serial no.	Inventory no.	Description
BionticsAstra	77483	CP8FD2		1334	4271	Press from the F series for low-maintenance production

MDC CONTROL

Controller Name
CP8FD2_controller

Type
MDE_UDP_CTRL_01

Machine address

Machine port

Log telegrams
0

Send immediately
false

SIGNALS

SIGNAL	TYPE
DONE	I

DNC CONTROL

Upload timeout (sec)
0

Download timeout (sec)
0

Automatic deletion
false

Plugin for machine configuration
MAZAK

Fig. 19: Dialog for configuring an asset in EDGE CONNECT

To add an asset:

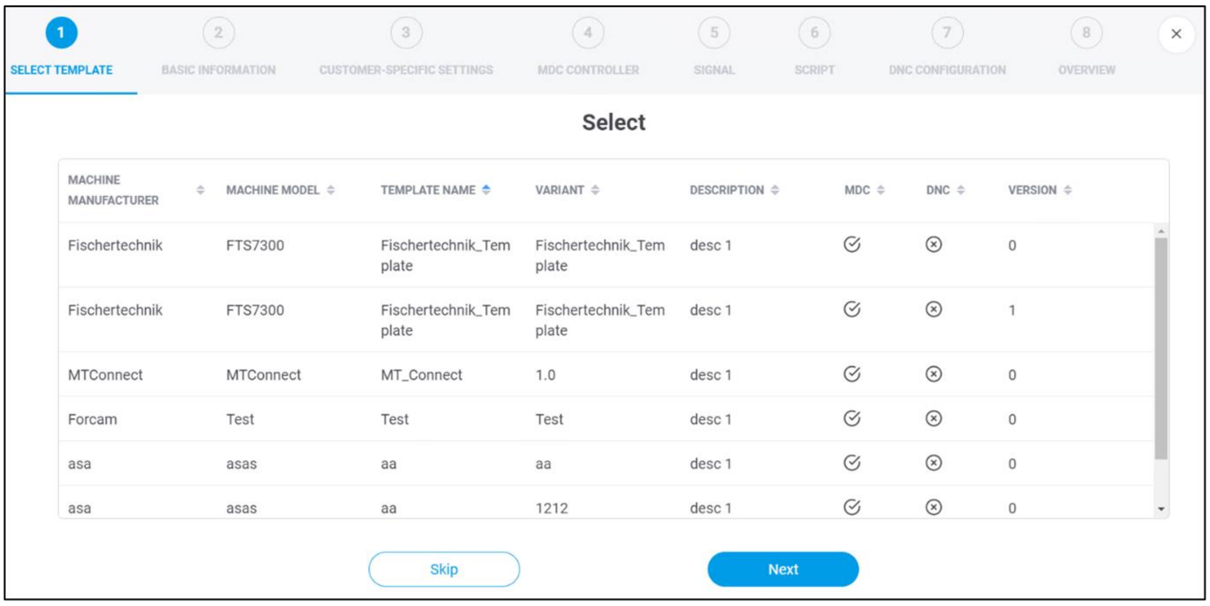
1. Click on **+ Asset** in the node details.
   ➔ The subsequent dialog guides you through the following eight steps for configuring an asset.

6.3.1 ① Select template

In EDGE CONNECT, multiple assets of the same type do not have to be completely reconfigured every 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 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 template **VERSION** indicates which version it is in. 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 asset connection will start with step ②.



Select

MACHINE MANUFACTURER	MACHINE MODEL	TEMPLATE NAME	VARIANT	DESCRIPTION	MDC	DNC	VERSION
Fischertechnik	FTS7300	Fischertechnik_Template	Fischertechnik_Template	desc 1	✓	⊗	0
Fischertechnik	FTS7300	Fischertechnik_Template	Fischertechnik_Template	desc 1	✓	⊗	1
MTConnect	MTConnect	MT_Connect	1.0	desc 1	✓	⊗	0
Forcam	Test	Test	Test	desc 1	✓	⊗	0
asa	asas	aa	aa	desc 1	✓	⊗	0
asa	asas	aa	1212	desc 1	✓	⊗	0

Fig. 20: Add asset - select template

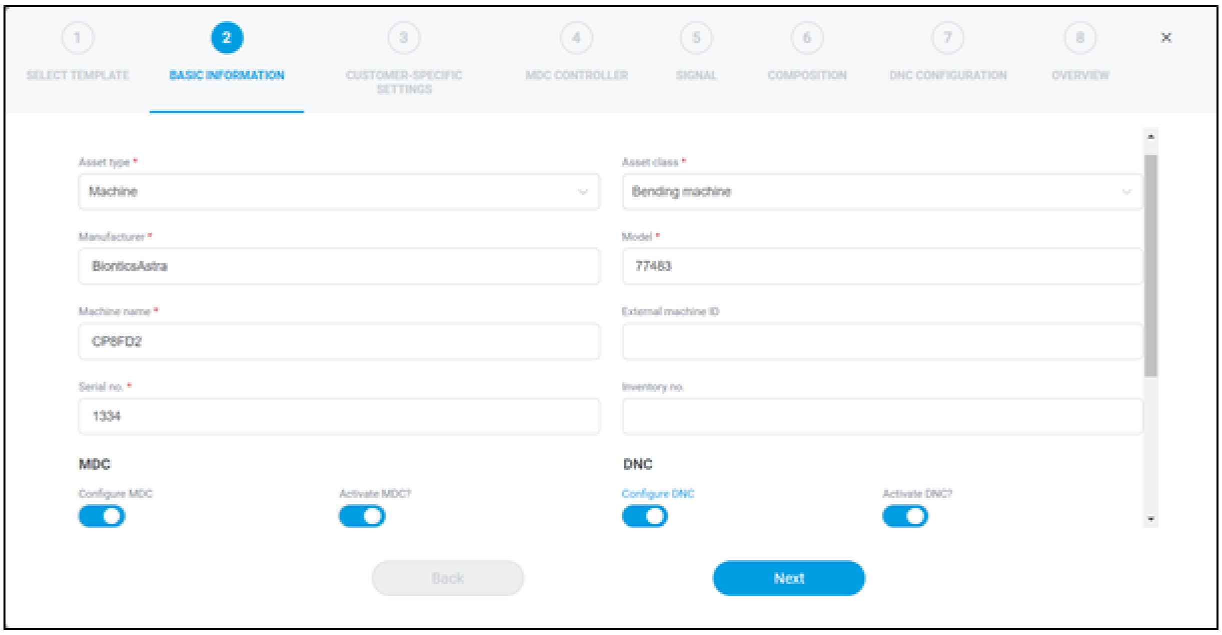
1. In the list, select the desired template for connecting the asset.
2. Click **Next**.

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

6.3.2 ② Basic information

This is where you enter basic information about the asset to be configured, 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.



Asset type *: Machine
Asset class *: Bending machine
Manufacturer *: BionticsAstra
Model *: 77483
Machine name *: CP8FD2
External machine ID:
Serial no. *: 1334
Inventory no.:

MDC
Configure MDC: on
Activate MDC?: on

DNC
Configure DNC: on
Activate DNC?: on

Fig. 21: Add asset - Basic Information

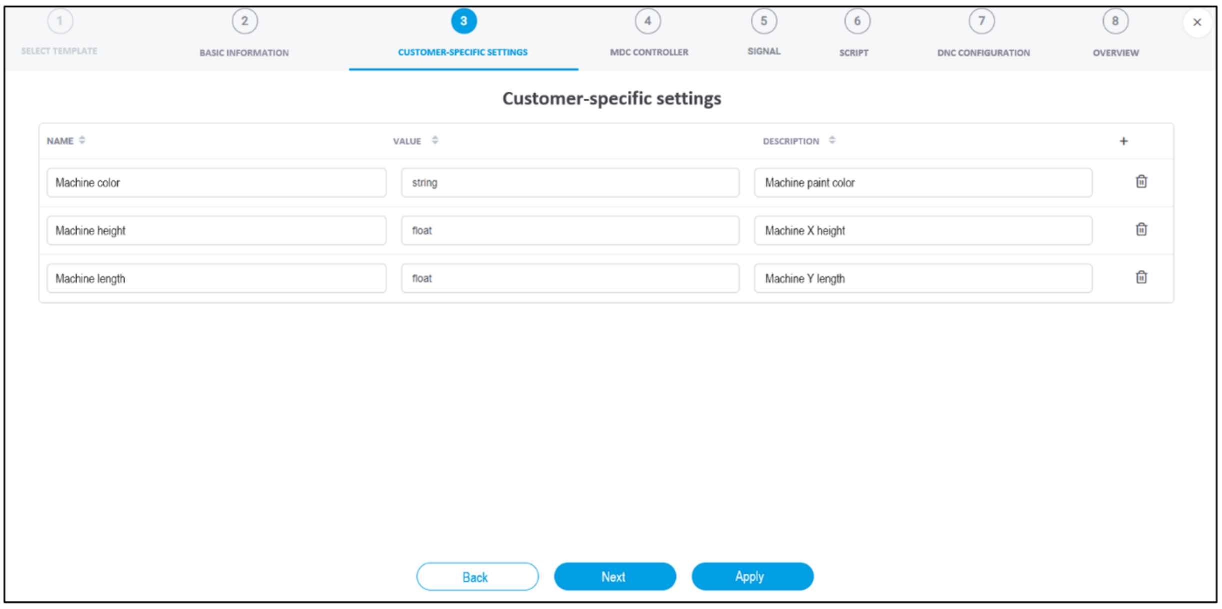
1. Select **asset type** and **asset classification**.
2. Enter **manufacturer**, **model**, **asset name** and **serial number**.
3. Optional: Enter additional information as desired.
4. Select the **Configure MDC** and/or **Configure DNC** check box.
5. Optional: Determine whether MDC or DNC should be activated initially.
6. Optional: Activate Data Lake.
7. Click **Next**.

6.3.3 ③ Customer-specific settings

This 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.



Customer-specific settings

NAME	VALUE	DESCRIPTION
Machine color	string	Machine paint color
Machine height	float	Machine X height
Machine length	float	Machine Y length

Fig. 22: Add asset – Customer-specific settings

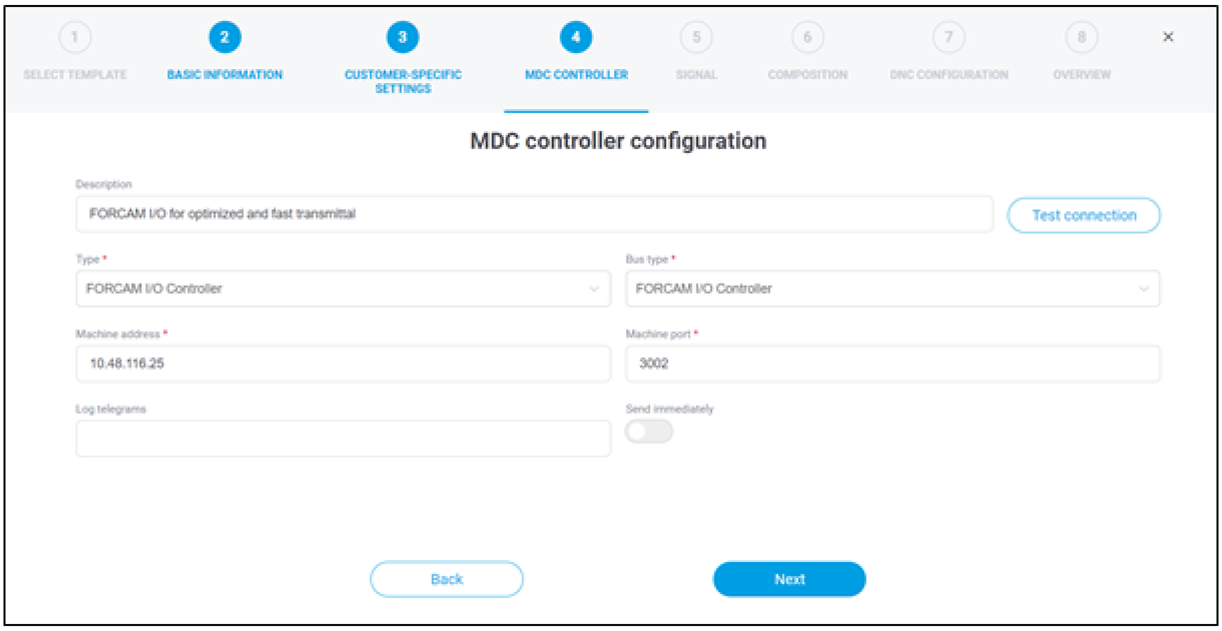
1. Click on the **+** on.
2. Enter the necessary parameters.
3. Click **Next**.

6.3.4 ④ MDC Controller

Option to configure an MDC controller. Specifies the type and method of connection to the asset. FORCAM supports all common controllers on the market and continuously strives to expand their availability. An overview of the current FORCAM plugins is listed in section 8.3.

The bus type is a specific communication protocol of the controller type. Many controllers have only one protocol and therefore only one bus type to choose from (e.g., bus type **FORCAM I/O Box connection** for controller **FORCAM I/O Box**). For the Siemens S7 controller, for example, several protocols are possible, which is why there are several bus types available.

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



MDC controller configuration

Description: FORCAM I/O for optimized and fast transmittal

Type *: FORCAM I/O Controller

Bus type *: FORCAM I/O Controller

Machine address *: 10.48.116.25

Machine port *: 3002

Log telegrams:

Send immediately: off

Fig. 23: Add asset - MDC controller

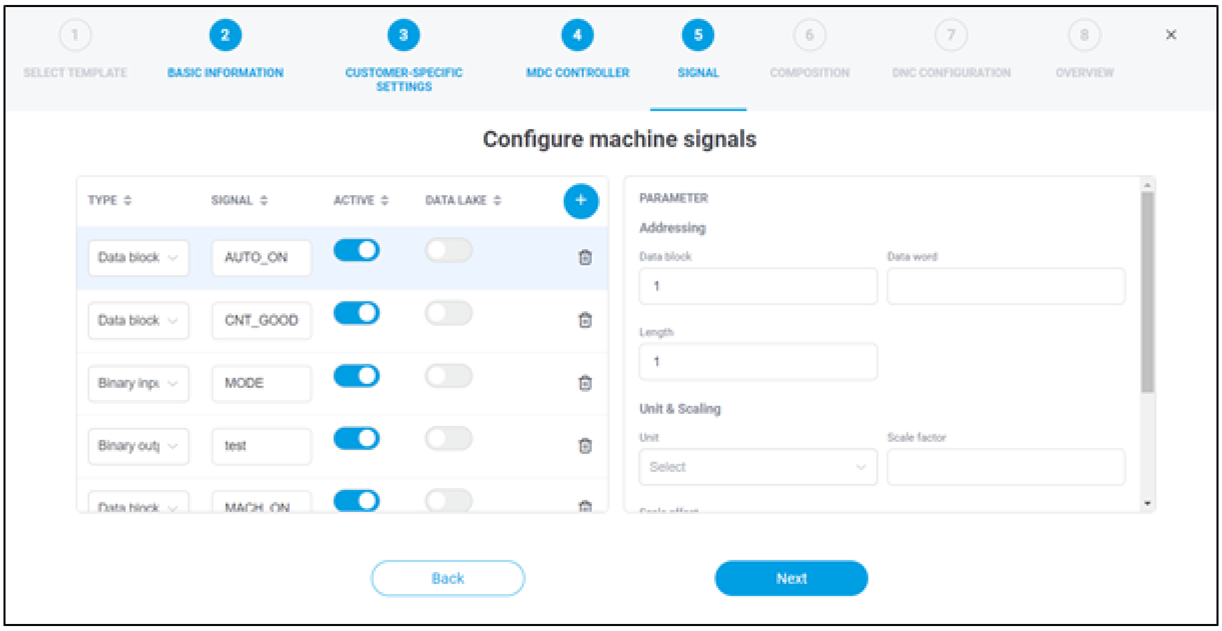
1. Optional: Enter a description of the controller.
2. Select the controller **Type**.
   ➔ Additional configuration parameters appear depending on the selected type.
3. Select the **Bus type**.
   The possible selection depends on what type of control was previously selected.
4. Adjust the configuration according to the controller.
5. Optional: Check the connection to the machine.
6. Click **Next**.

ⓘ The **Log telegrams** function is used to log out of the UDP telegrams in the DCU log. The number of to be logged characters 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 control (step ④), different listings of the signal types are displayed. The Data Lake can be used to record and save all data. Data Lake storage can be switched on and off per signal. Units can be recorded on individual signals (e.g., Degrees Celsius or liters per minute), and scaling factors can be defined.

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



Configure machine signals

TYPE	SIGNAL	ACTIVE	DATA LAKE
Data block	AUTO_ON	●	○
Data block	CNT_GOOD	●	○
Binary inp	MODE	●	○
Binary out	test	●	○
Data block	MACH_ON	●	○

PARAMETER

Addressing

Data block: 1
Data word:
Length: 1

Unit & Scaling

Unit: Select
Scale factor:

Fig. 24: Add asset – Signals

1. Click on the **+** icon.
2. Select the **type**, enter the **signal name** and optionally activate the **Data Lake** switch.
3. Specify plugin-specific **signal parameters**.
4. Optional: Enter **units** & **scaling** and **description**.
5. Click **Next**.

6.3.6 ⑥ Composition

In this step, the recorded signals are interpreted, from which logical conclusions about the asset behavior can be drawn. As a result, measurement values, maintenance information and production states are available. In this step ⑥ **Composition**, conditions for the interpretation of the signals are defined. There are two ways to enter these conditions: In the **SCRIPT** section, text-based code is displayed and edited (see Fig.26), whereas graphical blocks can be used in the **GRAPHIC** section (see Fig.25). These 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 commands even if you are new to programming in general. 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.

⚠ If you use list a list, don't forget to empty this list.

⚠ It is not possible to switch from SCRIPT to GRAPHIC mode. If the graphic was converted to a script, the script can be edited in the SCRIPT area but it cannot be reset to the block variant.

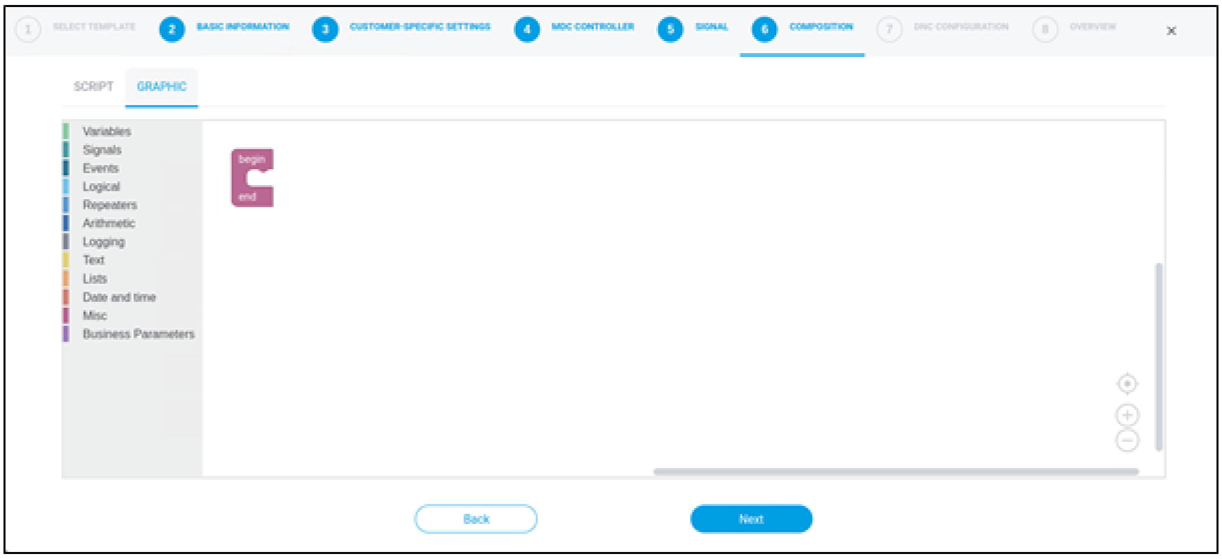
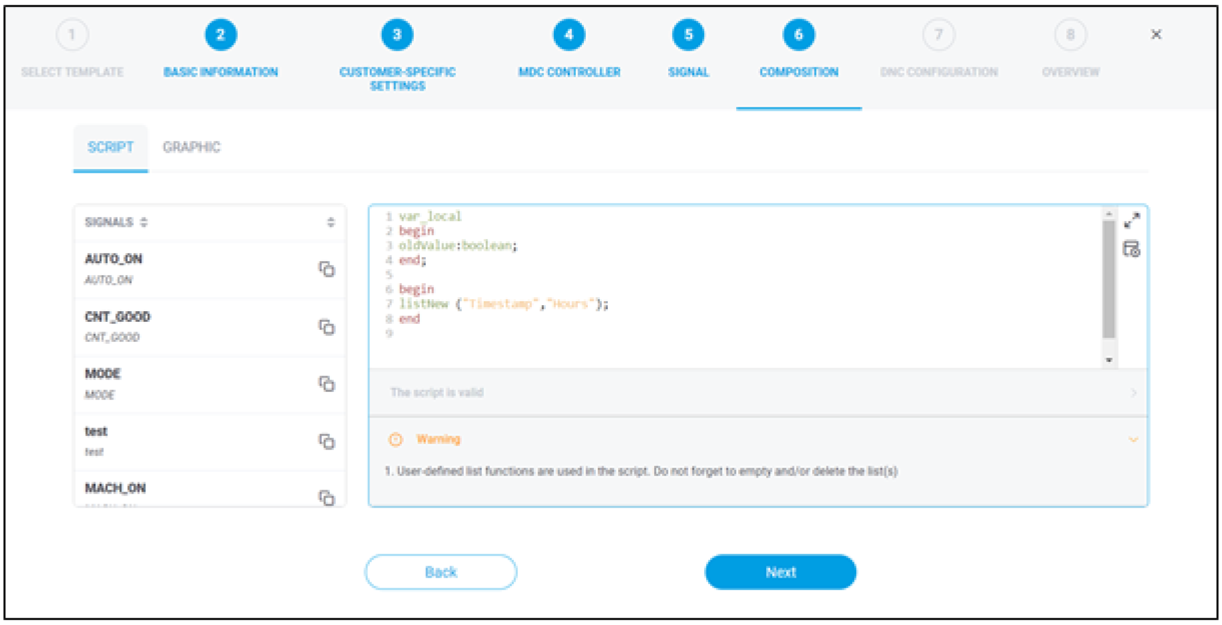


Fig.25: Add asset– GRAPHIC mode

ⓘ The annex of this manual contains sample scripts and script functions (see sections 8.5 and 8.6).

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



```
var_local
begin
oldValue:boolean;
end;

begin
listNew ("Timestamp","Hours");
end
```

The script is valid

Warning

1. User-defined list functions are used in the script. Do not forget to empty and/or delete the list(s)

Fig.26: Add asset - SCRIPT

1. Enter the desired script into the central input field.
2. Optional: Check the script’s validity at the top right of the **SCRIPT** screen.
3. Click **Next**.

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

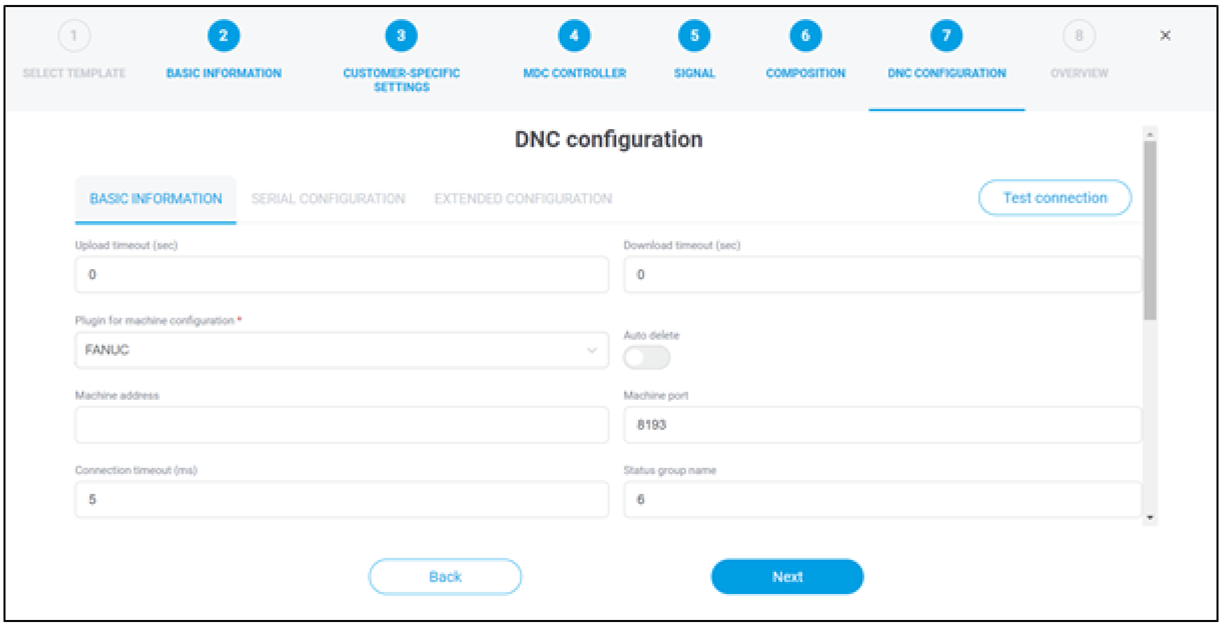
ⓘ The script editor can be set to full screen via the maximize icon.

6.3.7 ⑦ DNC Configuration

Option to configure a DNC controller. Specifies the type and method of transferring an NC file to the asset.
FORCAM supports all common controllers on the market and continuously strives to expand their availability. An overview of the current FORCAM plugins is listed in section 8.3.

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

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



DNC configuration

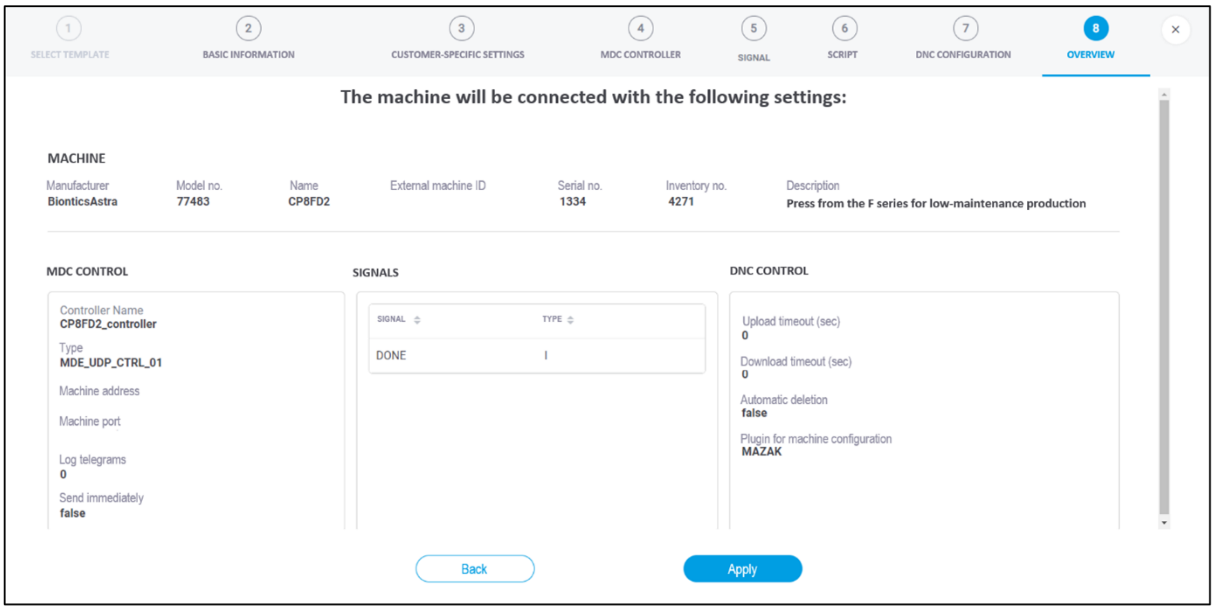
Upload timeout (sec): 0	Download timeout (sec): 0
Plugin for machine configuration *: FANUC	Auto delete
Machine address:	Machine port: 8193
Connection timeout (ms): 5	Status group name: 6

Fig. 27: Add asset - DNC configuration

1. Optional: Enter the **Upload** and **Download timeout**.
2. Select the **plugin for asset configuration**.
   ➔ Additional configuration parameters may appear in further tabs, depending on the selected plugin.
3. Optional: Set automatic deletion.
   If enabled, the NC file is automatically deleted from the asset after it has been read.
4. Enter additional configuration parameters in the remaining tabs, depending on the selected plugin.
5. Verify the connection to the asset.
6. Click **Next**.

6.3.8 ⑧ Overview

A summary of the previous configuration from all steps and a list of 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.16).



The machine will be connected with the following settings:

MACHINE

Manufacturer	Model no.	Name	External machine ID	Serial no.	Inventory no.	Description
BionticsAstra	77483	CP8FD2		1334	4271	Press from the F series for low-maintenance production

MDC CONTROL

Controller Name
CP8FD2_controller

Type
MDE_UDP_CTRL_01

Machine address

Machine port

Log telegrams
0

Send immediately
false

SIGNALS

SIGNAL	TYPE
DONE	I

DNC CONTROL

Upload timeout (sec)
0

Download timeout (sec)
0

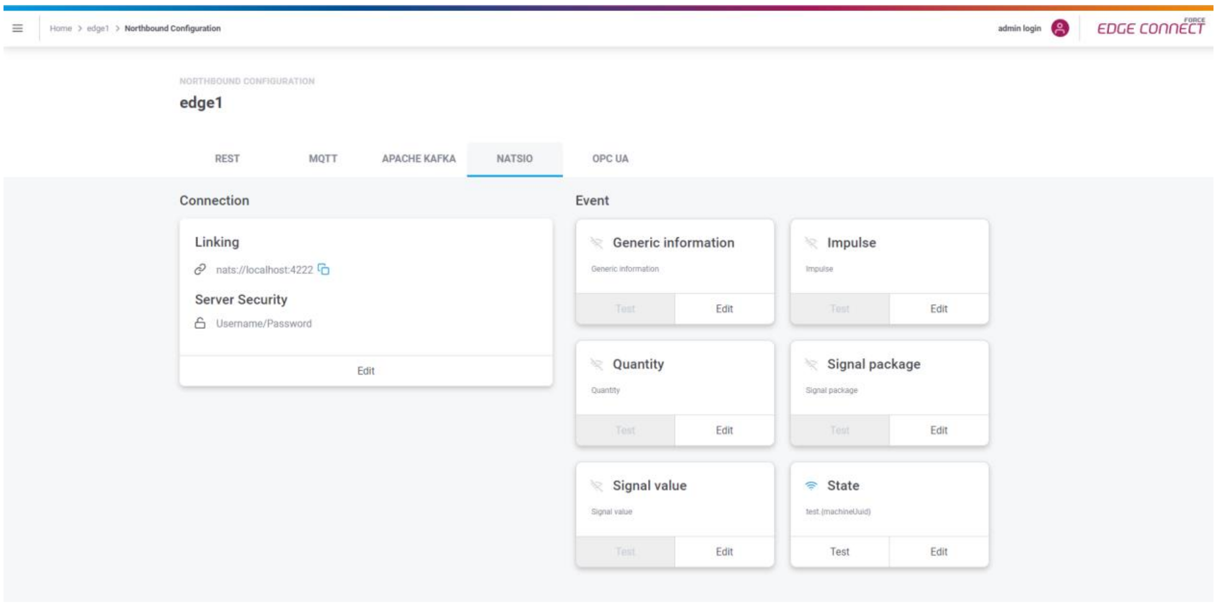
Automatic deletion
false

Plugin for machine configuration
MAZAK

Fig. 28: Add asset - 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.



Linking
nats://localhost:4222
Server Security
Username/Password

Generic information — Generic information
Impulse — Impulse
Quantity — Quantity
Signal package — Signal package
Signal value — Signal value
State — test.{machineUuid}

Fig. 29: Northbound configuration in EDGE CONNECT

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

Under **PAYLOAD** the JSON body defines what 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: $unit$
}
```

Script functions allow events to use **Placeholders** (wildcards) which can be used to transfer different types of information. This can be used, for example, to get the machine ID or the time stamp formatted in UTC. Chapter 8.6 lists and explains all available script functions.
If **ACTIVE** 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 & 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.

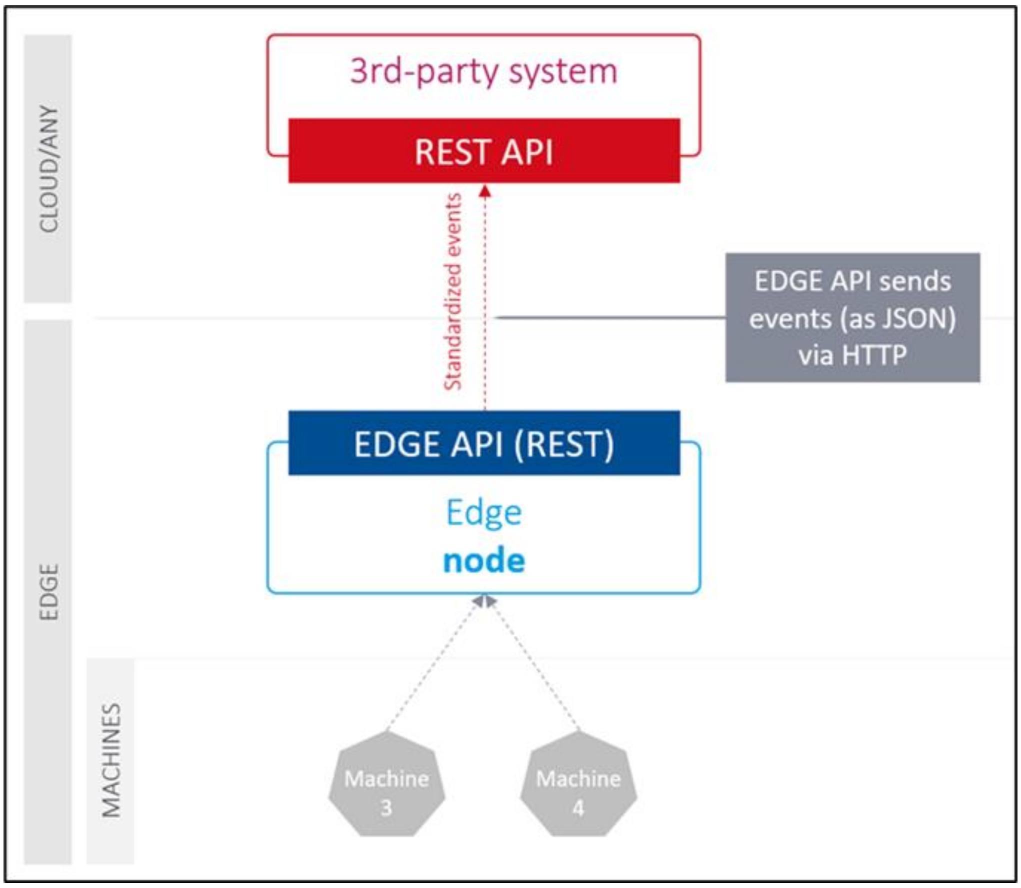


Fig. 30: Communication with superordinate systems via HTTP/REST

MQTT messaging

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

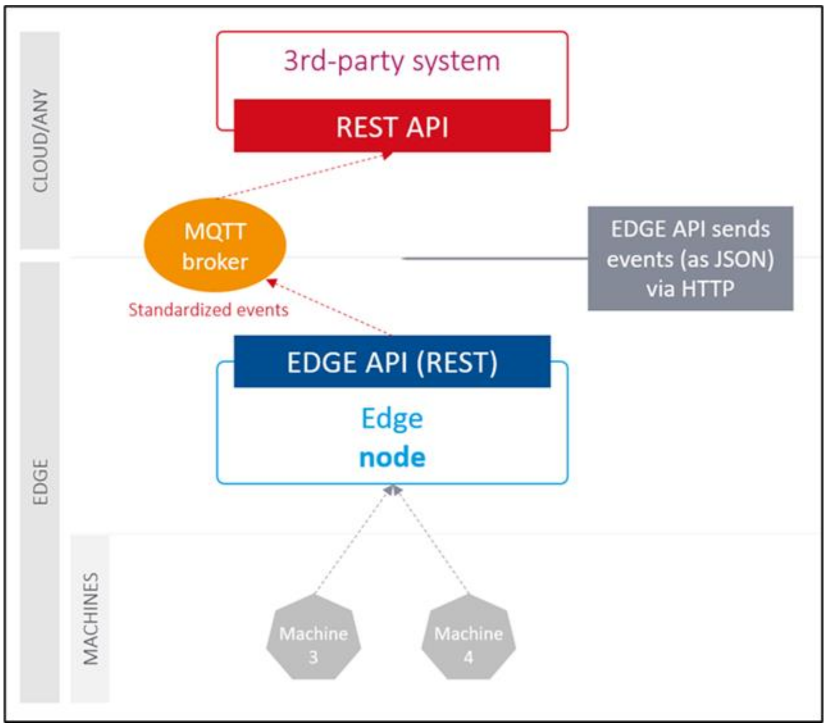


Fig. 31: Communication with superordinate systems via MQTT broker

Apache Kafka

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

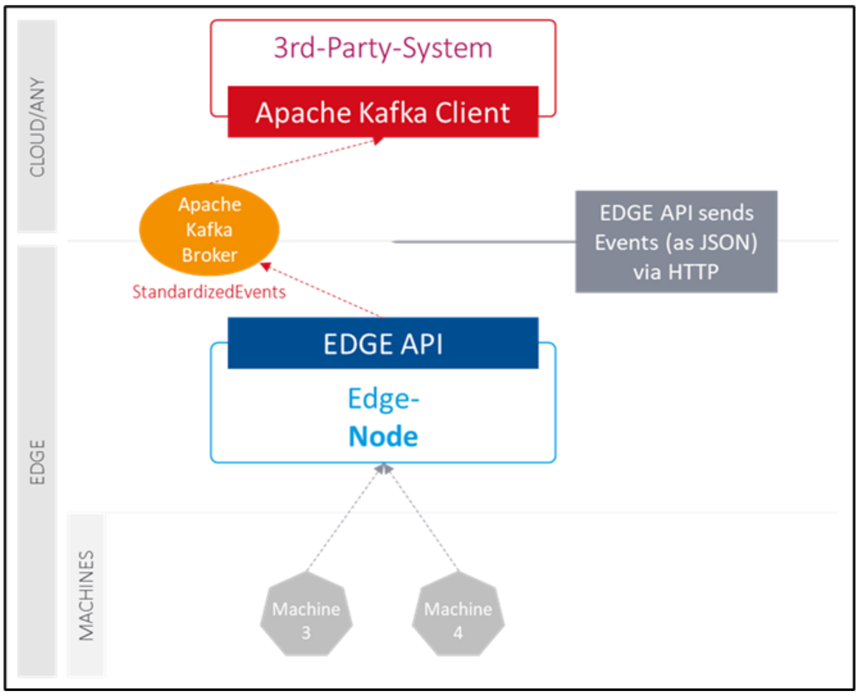
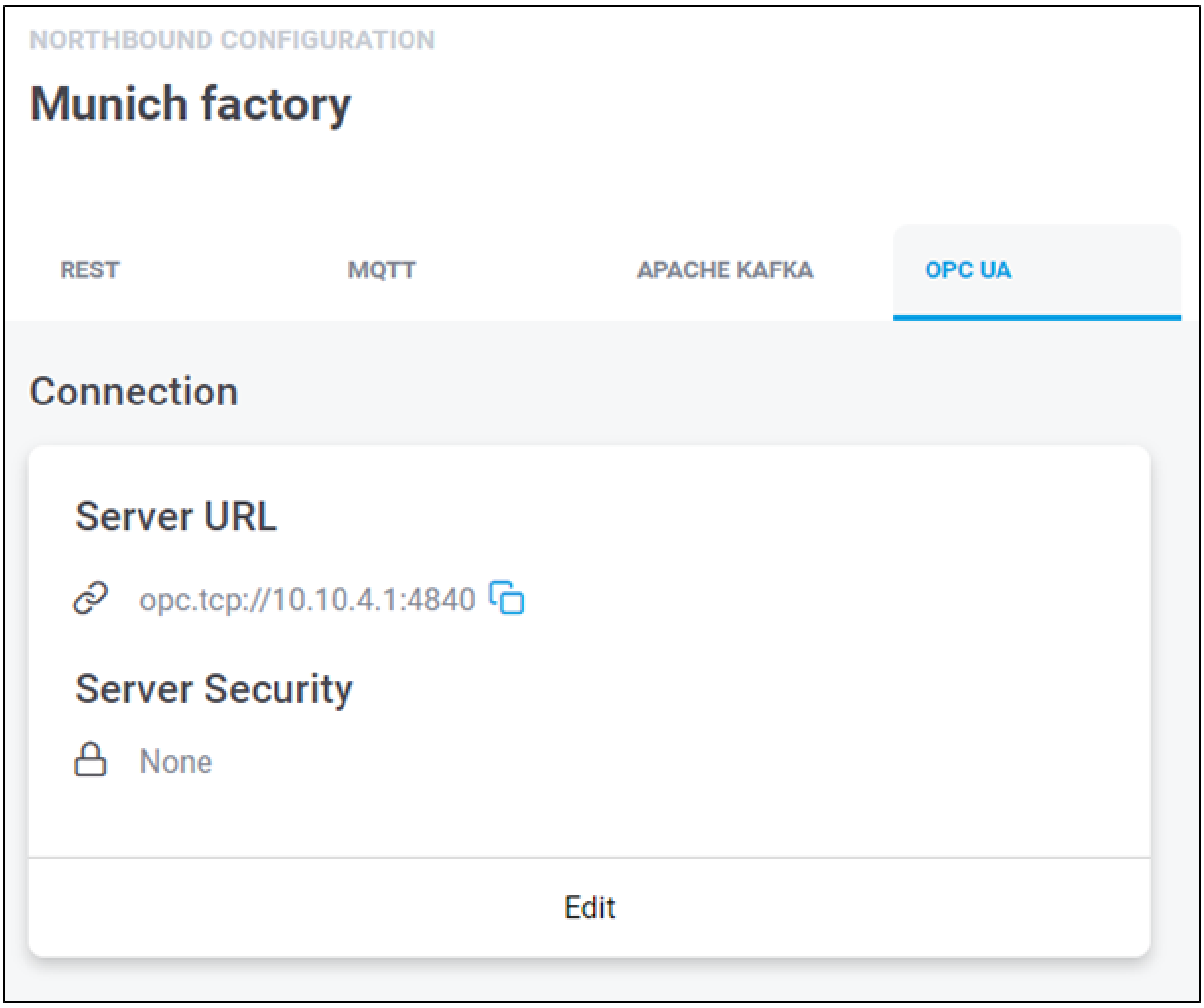


Fig. 32: Communication with superordinate systems via Apache Kafka

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 connects 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. 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.



Munich factory

Connection

Server URL

opc.tcp://10.10.4.1:4840

Server Security

None

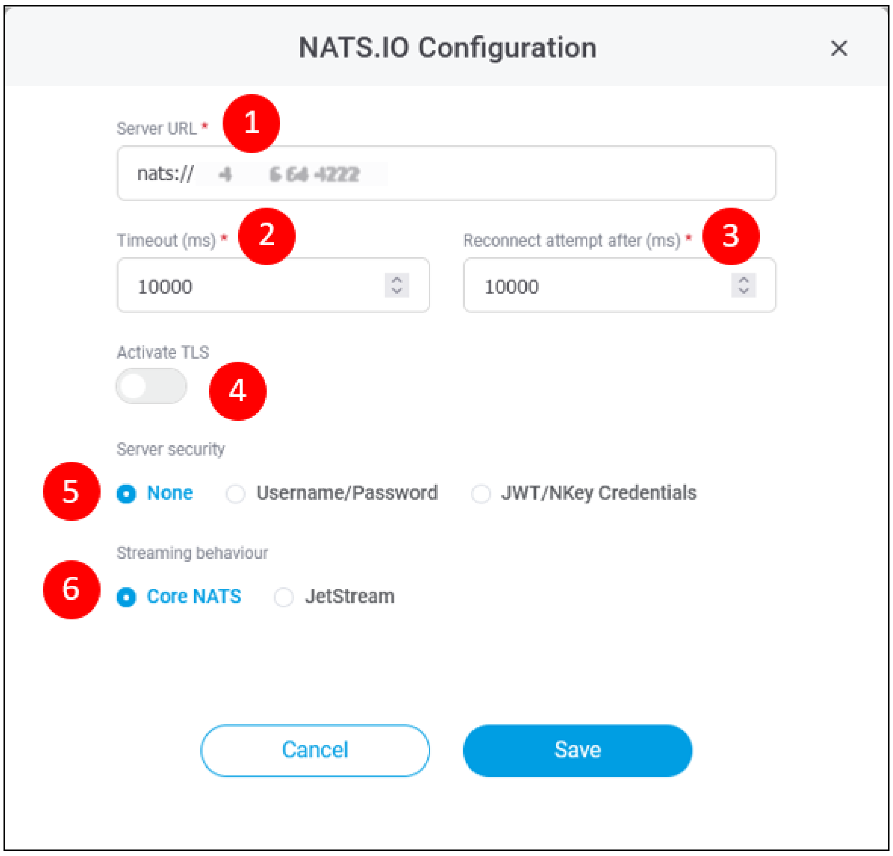
Fig. 33: Event Configuration via OPC UA

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).

NATS Core as well as 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.

We generally recommend creating one subject per asset and to use a customized script for optimum integration of the subject.


NATS.IO Configuration

Server URL * (1)
nats:// [redacted]

Timeout (ms) * (2)
10000

Reconnect attempt after (ms) * (3)
10000

Activate TLS (4)

Server security (5)
[✓ None] Username/Password JWT/NKey Credentials

Streaming behaviour (6)
[✓ Core NATS] JetStream

Fig. 32: Configuration of the NATS.io connection

(1) Server URL: Text field to enter the server URL
(2) Timeout (ms): Limits the time for connection attempts to the server
(3) Reconnect attempt after (ms): Time to wait until the next attempt to reconnect to the server
(4) TLS activation: Activates SSL/TSL encryption
(5) Server Security
    a. None: No authentication
    b. Username/Password: Authentication via user credentials
    c. JWT/NKey Credentials: Authentication via certificate
(6) Streaming behavior:
    d. Core NATS: Sent files are not saved.
    e. JetStream: Sent files are saved. Thereby applies that the message is transmitted at least once (Quality of Service Level 1).

6.4.2 Data & documents from superordinate system to EDGE

Via the EDGE API, EDGE CONNECT can be supplied with data and documents.
Technically, the transmission of NC files is 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:

Table 2: Interfaces for transferring data and documents

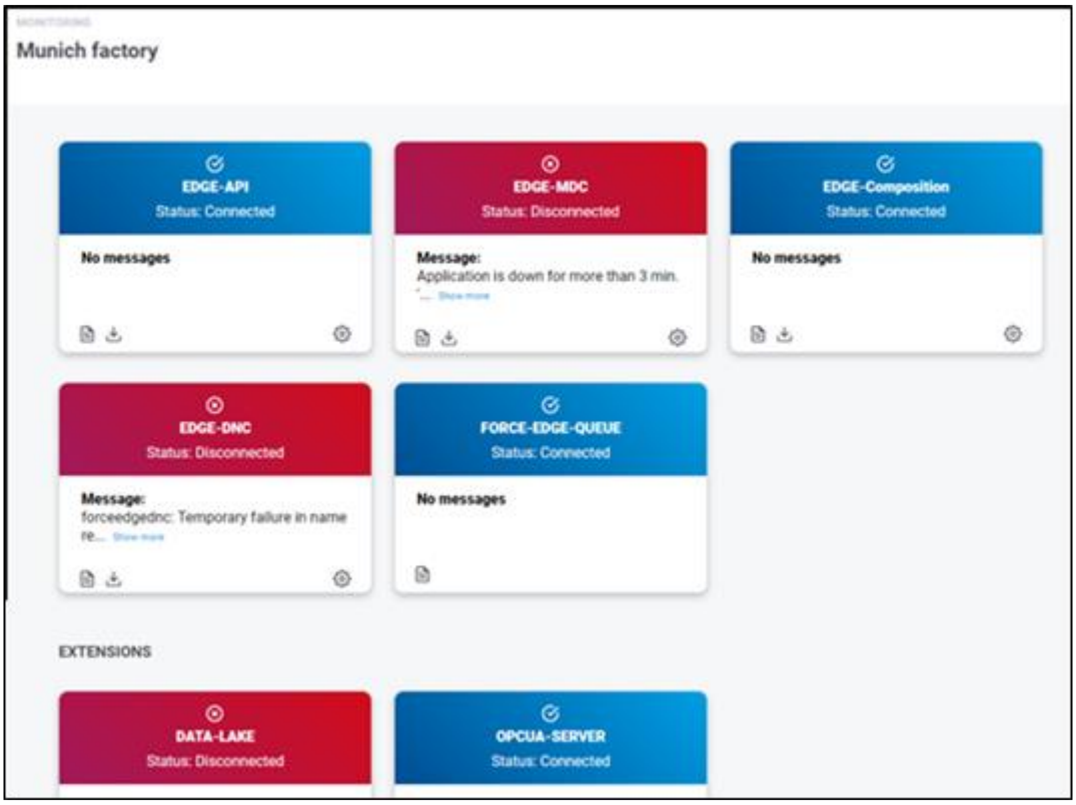
Interface	Description
Transfer of process and reference parameters	These business parameters can be used in the EDGE Composition Layer to supplement standardized events (e.g. order number or cycle time).
Transfer of signal parameters	Parameter values for specific signals can be transferred. These are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

6.4.3 Configuring an event

1. Click on the event configuration icon in the upper right part of the detailed view of a configured asset (see Fig.16).
2. In the subsequent dialog determine if **REST** or **MQTT** or **KAFKA** should be used.
3. In the upper bar, enter the **URL** and , optionally other details such as timeout, etc. Specifies where the events should be sent.
4. Set SSL certificate validation.
   If the **Check SSL certificate**switch is active, an application that does not have a valid or unsigned security certificate can also be connected to EDGE CONNECT via REST.
5. Select desired authentication and enter login data.
6. Configure events as desired.
7. Save.

7 Monitoring

EDGE CONNECT provides the option to monitor individual components and extensions 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 icon in the upper right area in the asset overview (see Fig.16).



MONITORING

Munich factory

EDGE-API
Status: Connected
No messages

EDGE-MDC
Status: Disconnected
Message:
Application is down for more than 3 min.
'.... Show more

EDGE-Composition
Status: Connected
No messages

EDGE-DNC
Status: Disconnected
Message:
forceedgednc: Temporary failure in name re.... Show more

FORCE-EDGE-QUEUE
Status: Connected
No messages

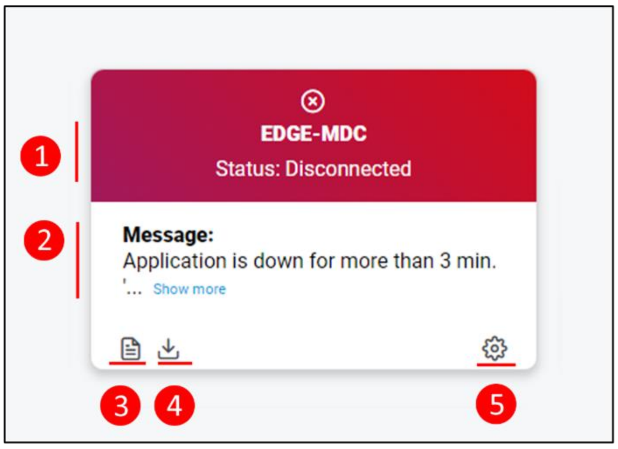
EXTENSIONS

DATA-LAKE
Status: Disconnected

OPCUA-SERVER
Status: Connected

Fig. 33: Monitoring in EDGE CONNECT

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



Message:
Application is down for more than 3 min.
'...

Fig. 34: "EDGE-DNC" component in the Monitoring page

(1) Current status of the component.
(2) Message in the event of an error.
Clicking **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 of a specific day.
(5) Setting the log-level.

8 Annex

8.1 Document conventions

Table 3: Fonts, formatting and characters used

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 ⓣ.

8.2 Abbreviations and terms used

Table 4: 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.

Abbreviation	Description
MDC	Machine Data Connection
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
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
UTC	Coordinated Universal Time
Wildcard	Placeholder for other characters

8.3 List of supported plugins

MDC Plugins

Table 5: List of all supported machine connection variants

Name	Read	Write	Transmission type Polling/Event-based
AUDI SPS	X	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		
Heidenhain	X	X	X/
MAKINO Pro 3/Pro 6	X		
Mazak	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		
Omron	X		
OPC Classic	X	X	X/
OPC UA	X	X	/X
OPC XML	X		X/
Rockwell/Allen Bradley	X	X	X/
Siemens LOGO	X	X	X/

Name	Read	Write	Transmission type Polling/Event-based
Siemens S5 with CP	X		
Siemens S5 without CP	X		
Siemens S7 with CP	X	X	X/
Siemens S7 without CP	X	X	X/
SQL Database Exchange	X		X/
Weihenstephan	X		X/
Wiesemann & Theis (WUT)	X		X/

DNC Plugins

Table 6: List of all supported NC machine connection variants

Name	Read	Write
COM	X	X
Heidenhain	X	X
Mazak DNC	X	X
RPC Plugin	X	X
FTP Plugin	X	X
FANUC	X	X
File Handler (File Copy)	X	X
File Handler Server	X	X
MOXA Box	X	X

8.4 Standardized events

Table 7: Events and their function in EDGE CONNECT

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	State (production or downtime)

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

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).	-

8.7 Table of figures