

---

# **MORPH.pro** **SMARTUNIFIER**

## **SMARTUNIFIER User Manual**

*Release 1.2.0*

**Amorph Systems GmbH**

**Jun 16, 2021**

# ABOUT SMARTUNIFIER

<b>1</b>	<b>About SMARTUNIFIER</b>	<b>2</b>
1.1	What is SMARTUNIFIER	2
1.2	What does SMARTUNIFIER do	3
1.3	Important Use Cases with SMARTUNIFIER	4
1.3.1	Anything-To-Anywhere IT Interface	4
1.3.2	Reusable Interfaces and Interface Models	5
1.3.3	Integrate Legacy Equipment	6
1.3.4	Implement Fab Communication Scenario	7
1.3.5	Provide Base for Remote Maintenance and Health Monitoring	8
1.3.6	Migrate to Industry 4.0	9
1.3.7	Allow Unlimited Scalability	10
1.3.8	Enable Internet of Things	11
1.4	Connectivity Endpoints and Data Formats	12
1.4.1	Connectivity Endpoints / Communication Protocols	13
1.4.2	Data Formats	15
1.5	What has changed in 1.2.0	15
<b>2</b>	<b>How to integrate with SMARTUNIFIER</b>	<b>17</b>
2.1	Information Models	17
2.1.1	What are Information Models	17
2.1.2	How to create a new Information Model	17
2.1.3	Node Types	19
2.1.4	Data Types	25
2.2	Communication Channels	26
2.2.1	What are Channels	26
2.2.2	How to create a new Channel	26
2.2.3	Channel Types and Configuration	28
2.3	Mappings	46
2.3.1	What are Mappings	46
2.3.2	How to create a new Mapping	46
2.3.3	How to create Rules	48
2.4	Device Types	56
2.4.1	What are Device Types	56
2.4.2	How to create a new Device Type	56
2.5	Instances	58

2.5.1	What are Instances . . . . .	58
2.5.2	How to create a new Instance . . . . .	58
<b>3</b>	<b>Deployment</b>	<b>61</b>
3.1	What is a Deployment . . . . .	61
3.2	Deploy Locally . . . . .	62
3.3	Deploy with Docker . . . . .	63
3.4	Deploy with AWS Fargate . . . . .	65
3.4.1	Prerequisites . . . . .	65
3.4.2	Architecture . . . . .	69
3.4.3	Planning the Deployment . . . . .	71
3.4.4	Deployment Steps . . . . .	72
3.5	How to deploy, run and operate a deployed Instance . . . . .	75
3.5.1	How to deploy an Instance . . . . .	75
3.5.2	How to run an Instance . . . . .	75
3.5.3	How to stop an Instance . . . . .	75
3.5.4	How to delete a Deployment of an Instance . . . . .	76
3.5.5	How to un-deploy an Instance . . . . .	76
3.6	How to monitor a deployed Instance . . . . .	76
<b>4</b>	<b>Administration</b>	<b>78</b>
4.1	Deployment Endpoints . . . . .	78
4.1.1	What are Deployment Endpoints . . . . .	78
4.1.2	Deployment Endpoints Types . . . . .	78
4.2	User Management . . . . .	81
4.2.1	About User Management . . . . .	81
4.2.2	Add a new user . . . . .	81
4.2.3	Edit a user . . . . .	83
4.2.4	Delete a user . . . . .	85
<b>5</b>	<b>Demonstration Scenarios</b>	<b>88</b>
5.1	File-based data - CSV to REST-Server . . . . .	88
5.1.1	Overview . . . . .	88
5.1.2	Information Model . . . . .	89
5.1.3	Communication Channel . . . . .	90
5.1.4	Mapping . . . . .	91
5.1.5	Device Type . . . . .	92
5.1.6	Instance . . . . .	93
5.1.7	Deployment . . . . .	95
5.2	File-based data - Insert JSON data in SQL-Database . . . . .	96
5.2.1	Overview . . . . .	96
5.2.2	Prerequisite . . . . .	96
5.2.3	Information Model . . . . .	96
5.2.4	Communication Channel . . . . .	97
5.2.5	Mapping . . . . .	99
5.2.6	Device Type . . . . .	100
5.2.7	Instance . . . . .	100
5.2.8	Deployment . . . . .	101

5.3	File-based data - XML, Database, and MQTT	102
5.3.1	Overview	102
5.3.2	Prerequisites	102
5.3.3	Information Model	103
5.3.4	Communication Channel	104
5.3.5	Mappings	107
5.3.6	Device Type	108
5.3.7	Instance	109
5.3.8	Deployment	109
<b>6</b>	<b>Getting Help</b>	<b>111</b>
6.1	Troubleshooting	111
6.1.1	Instance works abnormally or hangs/crashes	111
6.1.2	Manager works abnormally or hangs/crashes	112
6.2	FAQ	112

**Integrate perfectly your  
Production-IT using**

**SMARTUNIFIER**  
ADVANCED IT-INTEGRATION PLATFORM

**AMORPH.pro**  
**SMARTUNIFIER**

## ABOUT SMARTUNIFIER

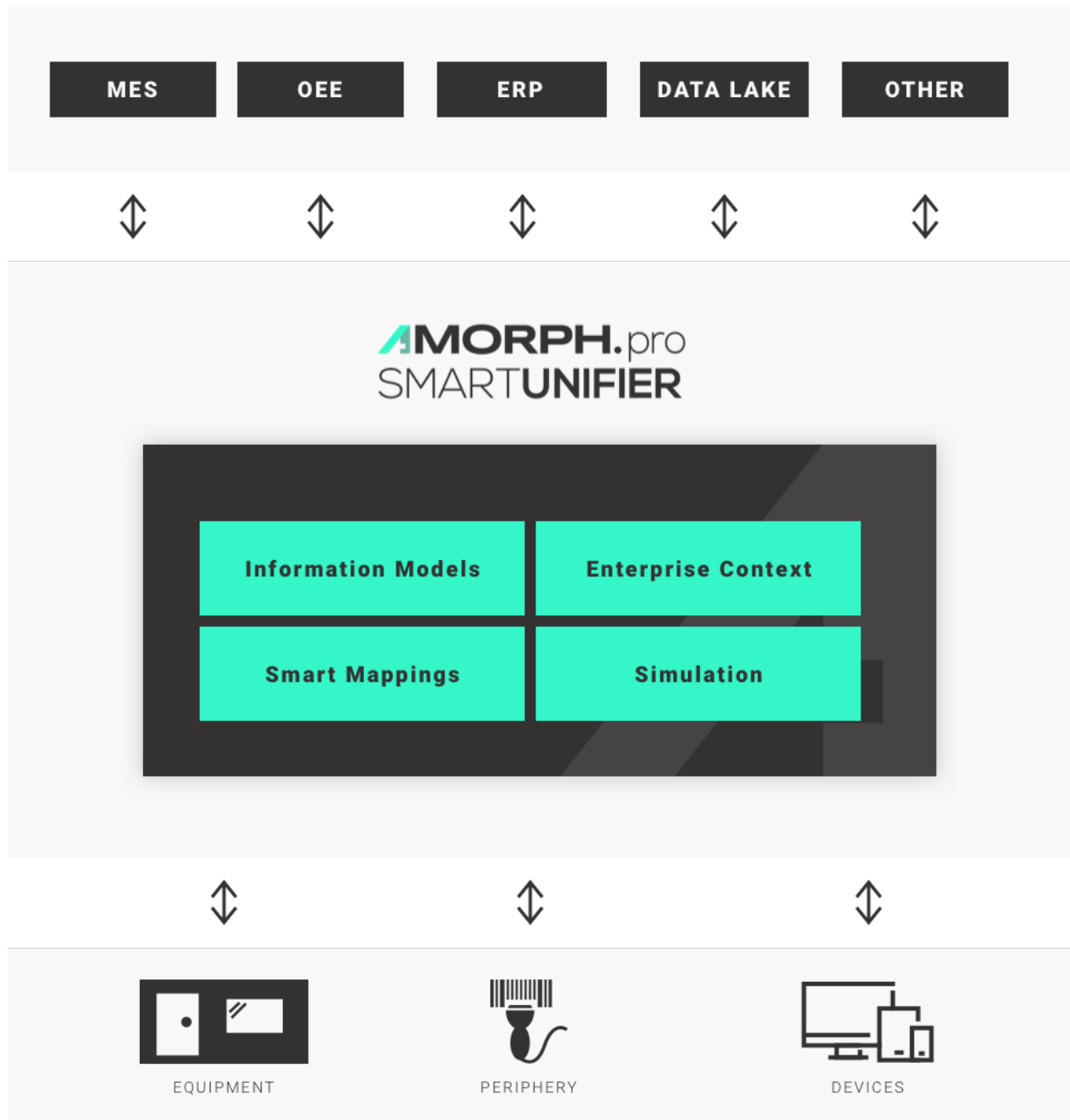
You are new to SMARTUNIFIER?

- Learn about the *SMARTUNIFIER* integration platform.
- Learn about the *connectivity use cases* you can address with SMARTUNIFIER.
- Check out the supported *connectivity endpoints and data formats*.
- Learn about *changes*.

### 1.1 What is SMARTUNIFIER

SMARTUNIFIER represents a powerful but very easy to use integration platform for interconnecting all industrial devices and IT systems including equipment, peripheral devices, sensors/actors, MES, ERP as well as cloud-based IT systems.

SMARTUNIFIER is the tool of choice for transforming data into real value and for providing seamless IT interconnectivity within production facilities.



## 1.2 What does SMARTUNIFIER do

- **SMARTUNIFIER** provides an easy way to collect data from any Data Source and is able to transmit this data to any Data Target.
- Data Sources and Data Targets (commonly referred to as Communication Partners) in this respect may be any piece of equipment, device or IT system, communicating typically via cable or Wi-Fi and using a specific protocol like e.g., OPC-UA, file-based, database, message bus.

- With SMARTUNIFIER several Communication Partners can be connected simultaneously.
- With SMARTUNIFIER it is possible to communicate unidirectional or bidirectional to each Communication Partner. i.e., messages and events can be sent and received at the same time.
- SMARTUNIFIER is able to translate and transform data to any format and protocol that is required by a certain Data Target. This includes different pre-configured protocols and formats, e.g., OPC-UA, file-based, database, message bus, Webservices and many direct PLC connections. In case a certain protocol or format is currently not available it can be easily added to SMARTUNIFIER.
- By applying so called Information Models, SMARTUNIFIER enables the same view to data regardless of the protocol or format being used to physically connect an equipment, device or IT system.
- A big advantage of SMARTUNIFIER is, that in many cases there is no need for coding when providing interfaces between different Communication Partners – providing a new interface is just drag and drop of data objects between data source(s) and destination(s).

## 1.3 Important Use Cases with SMARTUNIFIER

SMARTUNIFIER enables an easy and very efficient realization of many use cases that are crucial for gaining Industry 4.0 Excellence.

In the following subchapters some of the most important SMARTUNIFIER Use Cases are described. These give a comprehensive overview of the advanced SMARTUNIFIER Features.

### 1.3.1 Anything-To-Anywhere IT Interface

**Easy, fast and flexible bi-directional interconnection of multiple IT systems and equipment within a production facility.**

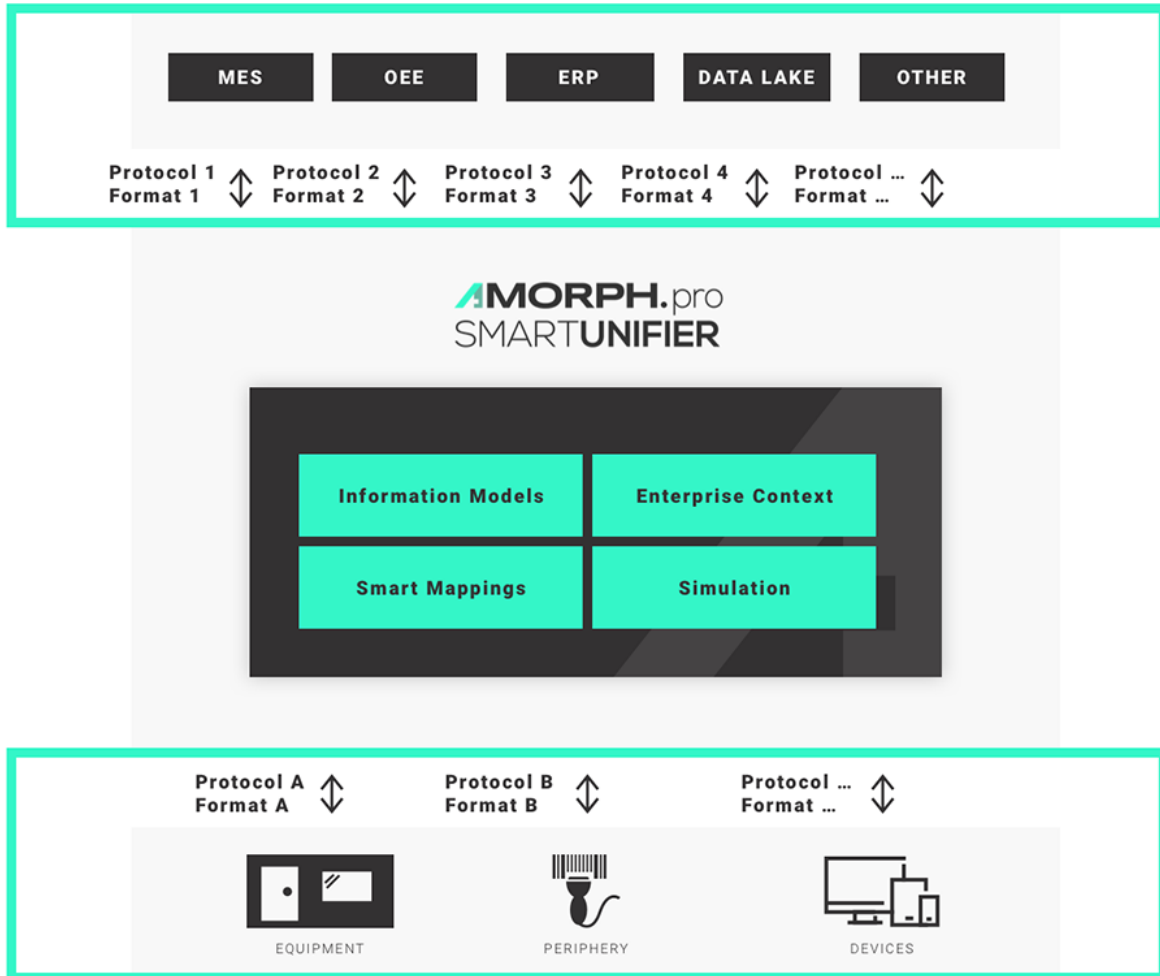
Interconnecting heterogeneous shop floor equipment and devices with IT systems and interconnecting different IT systems with each other is a central requirement for a successful transition to modern Industry 4.0 IT landscapes.

SMARTUNIFIER offers the unique capability to easily interconnect equipment and devices by allowing

- any number of parallel high-speed Communication Channels between equipment, devices and IT systems
- high-speed translation between different communication protocols and formats by applying configurable and reusable Information Models and Smart Mappings
- flexible integration of equipment periphery
- easy integration of enterprise-specific information (e.g., equipment -location/-name/-type/-capabilities) via configurable Enterprise Context
- riskless simulation of interfaces and communication scenarios



Results from renowned reference customers have shown that average equipment integration efforts and **cost can be reduced by up to 90%** using the SMARTUNIFIER and its advanced technologies to perform powerful IT integration by configuration instead of tedious interface programming.



### 1.3.2 Reusable Interfaces and Interface Models

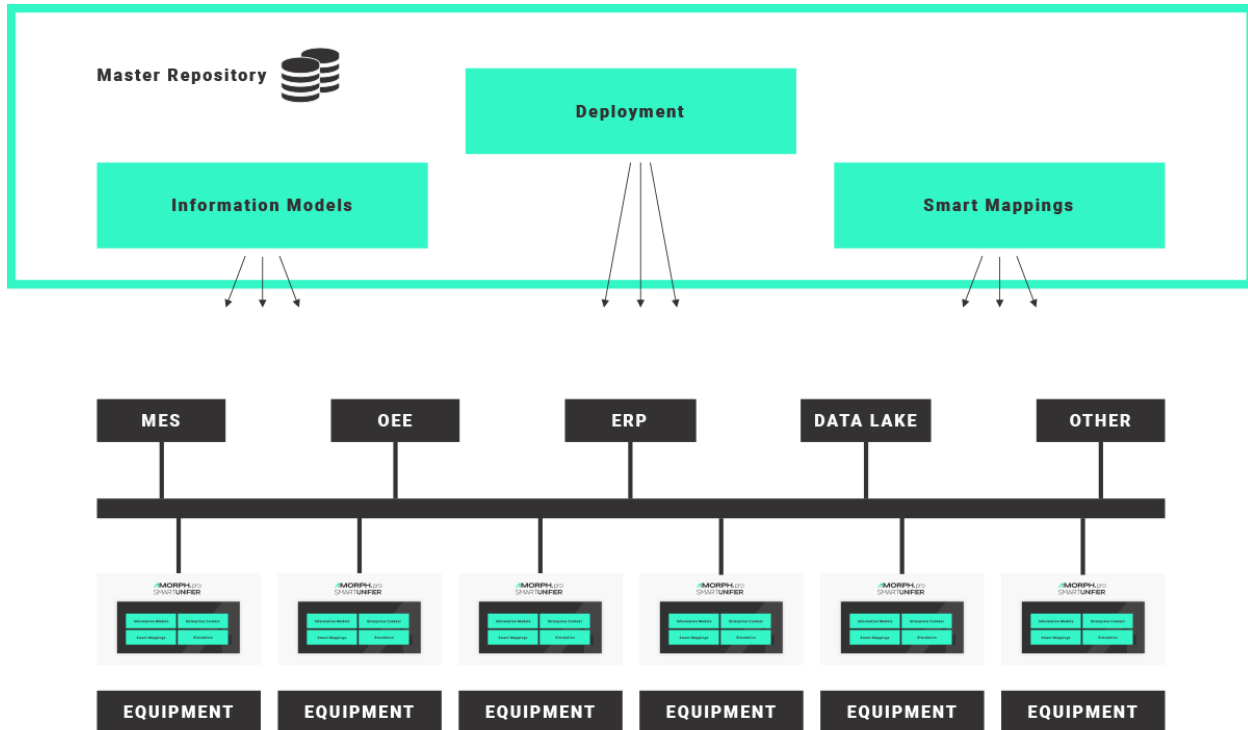
**Reuse interface configurations multiple times with minimum effort.**

When running an IT network with a higher number of installed SMARTUNIFIER Instances, all previously created interface configurations (Information Models and Smart Mappings) can be reused easily and shared across the whole installation. This way similar equipment types are integrated using the same connection and translation logic.

Changes and updates of interface configurations can be deployed from a centrally accessible Master Repository, eliminating the need to touch and update each equipment or device individually.

Summarized, SMARTUNIFIER allows a highly comfortable and effective management of very small to very large IT communication environments, creating minimum overhead and letting you reach

your main goal: Excellent Manufacturing with a full Industry 4.0 IT infrastructure.



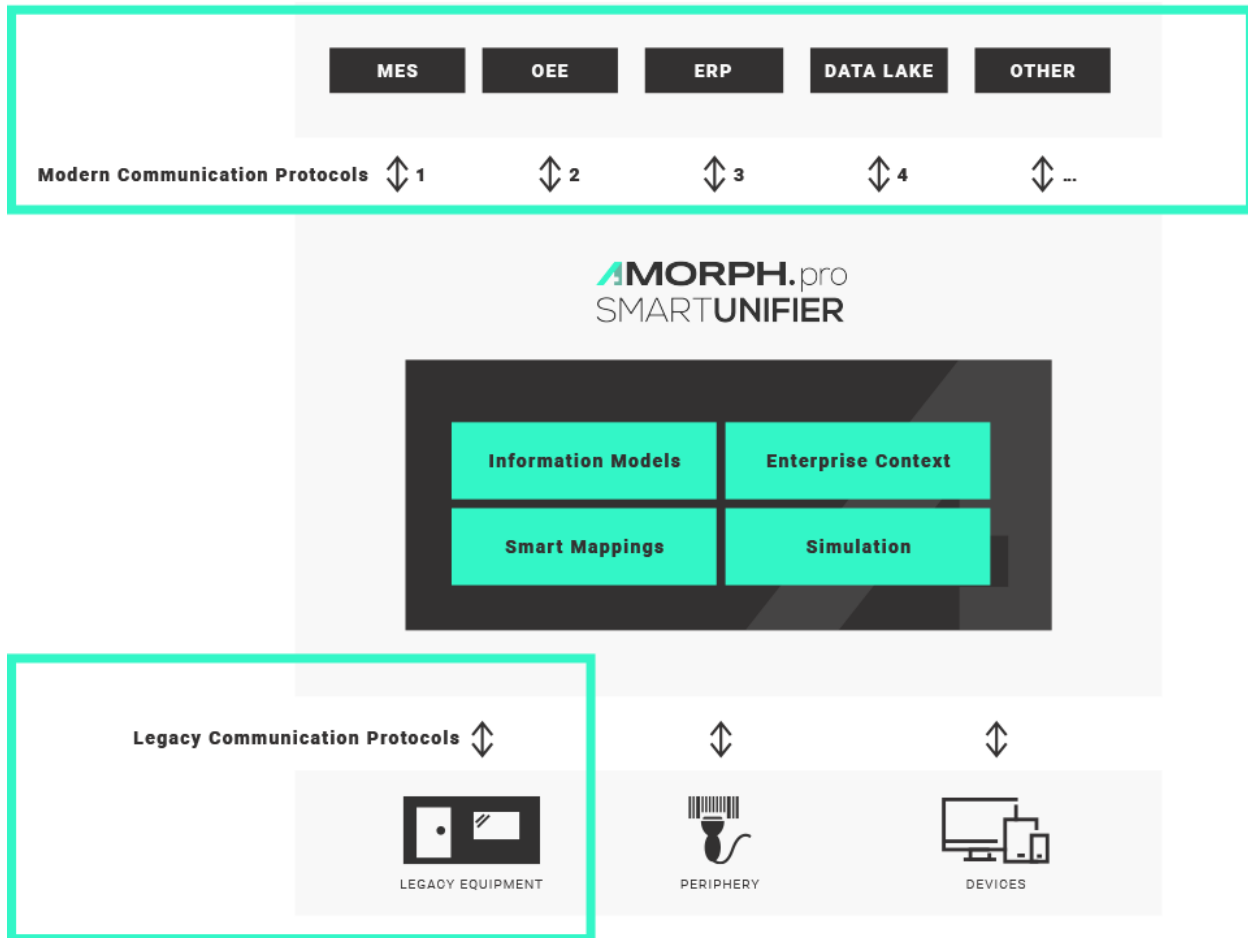
### 1.3.3 Integrate Legacy Equipment

**Fast adaptation of legacy communication protocols and formats to modern enterprise standards.**

By applying SMARTUNIFIER configurable protocol translation (Smart Mappings), modern communication standards like OPC-UA or XML over message bus are fully supported.

SMARTUNIFIER allows a really smooth migration from existing communication protocols and formats (e.g., between existing equipment and MES) to new Industry 4.0 standards.

This unique capability of SMARTUNIFIER is realized by simply using existing communication channels simultaneously with newly introduced channels. When finishing the migration, the old channels can be switched off without any risk.

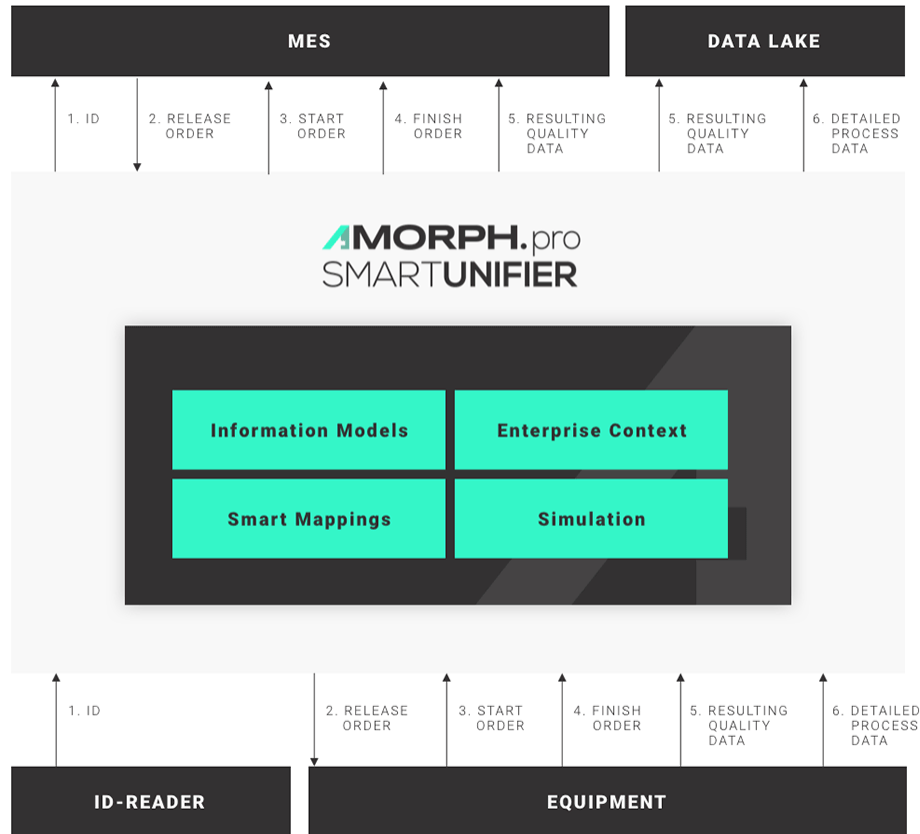


### 1.3.4 Implement Fab Communication Scenario

**Easily implement complete fab communication sequences that cover multiple steps.**

With SMARTUNIFIER it is not only possible to give access to simple equipment or device data and to provide „some data to MES and Cloud“, but also with SMARTUNIFIER complete communication scenarios between equipment to upper-level IT systems can be easily implemented.

The communication scenarios can cover all steps from identification, validation, order start as well as sending results and process data from equipment to MES or Cloud. Of course it is also possible to provide any parameter data (recipes) from MES or SCADA to equipment.



### 1.3.5 Provide Base for Remote Maintenance and Health Monitoring

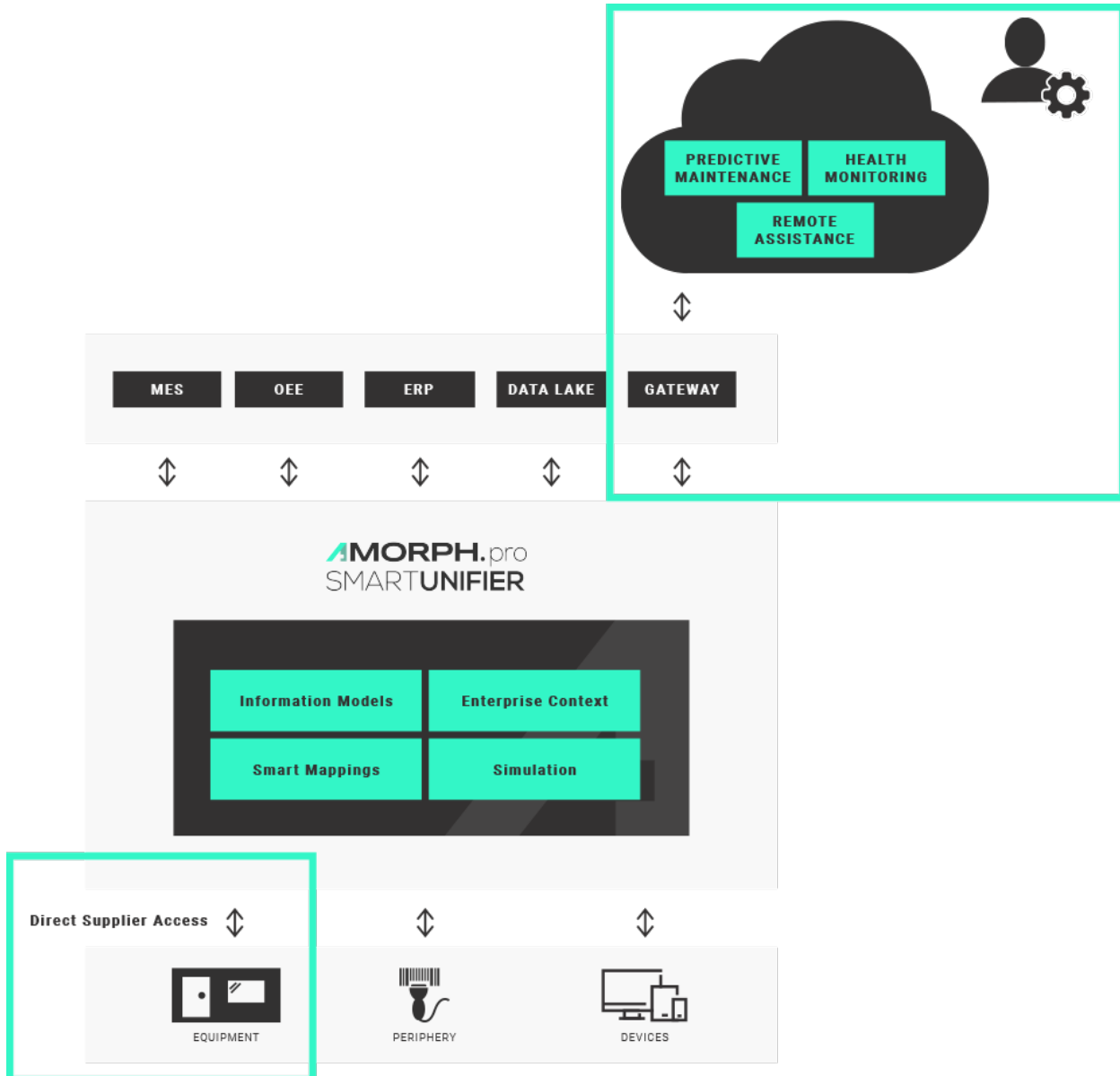
**Establish new services and business models by giving secured multi-channel access to equipment and device data in real-time.**

Production equipment can be integrated with SMARTUNIFIER to provide direct access for equipment suppliers or maintenance service providers to relevant equipment data (e.g., equipment status, equipment key parameters) via an equipment supplier's cloud infrastructure.

This way, new innovative business models for equipment suppliers are supported by building the base for "Production as a Service" offerings and remote predictive maintenance.

Also, further advanced business use cases with SMARTUNIFIER are possible, e.i., by implementing real-time equipment monitoring capabilities in a cloud environment.

Another SMARTUNIFIER use case is to give Remote Assistance to equipment suppliers in order to achieve production optimization and to ensure the most efficient usage of equipment resources for customers.



### 1.3.6 Migrate to Industry 4.0

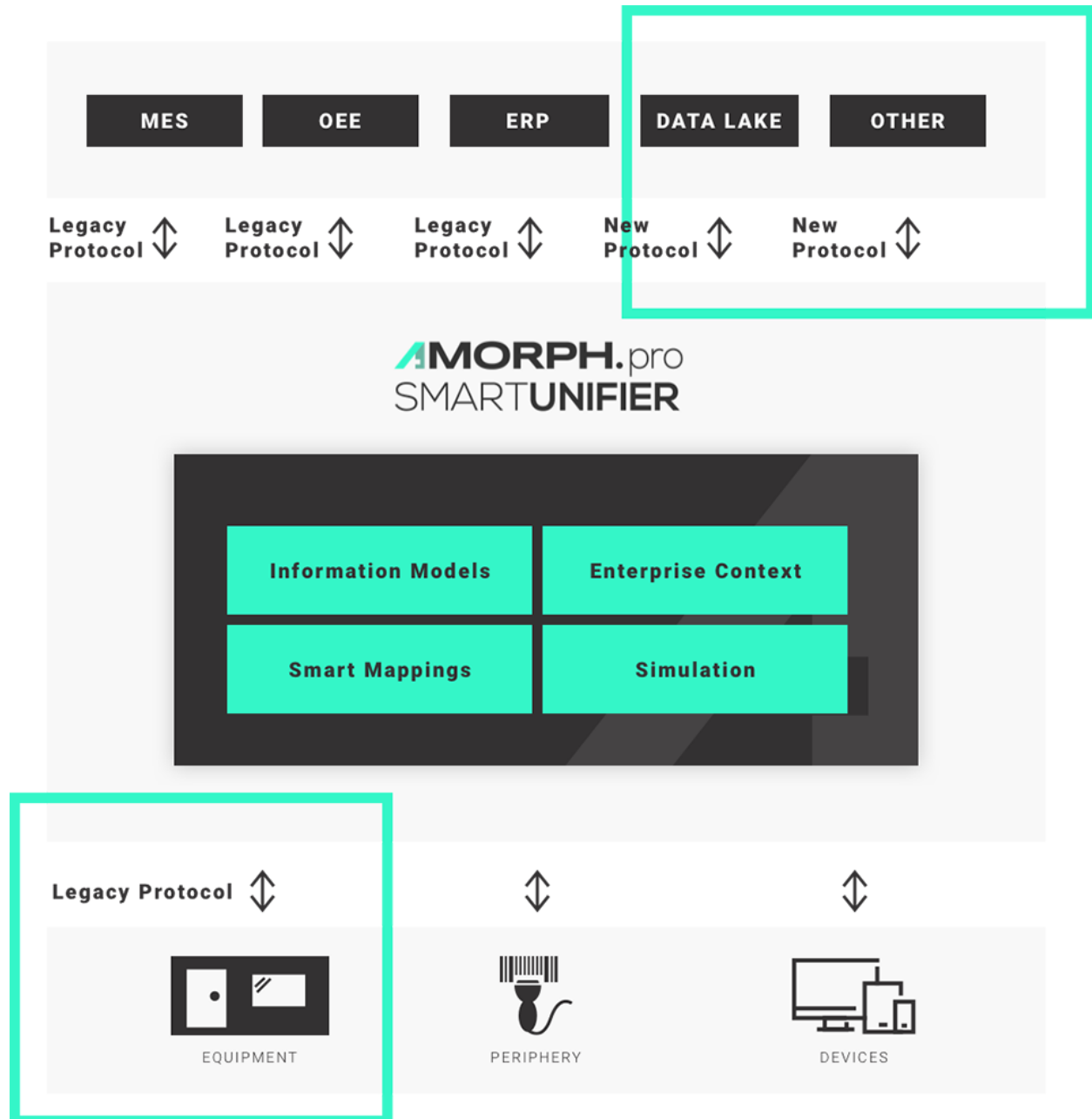
**Migrate step by step to modern communication standards and apply enterprise-wide semantics to data.**

A key feature of SMARTUNIFIER is to open an easy way to integrate new IT systems using modern communication protocols. This is realized by simply adding additional communication channels to the existing legacy channels.

Another feature of SMARTUNIFIER in this respect is, that all existing IT systems with their legacy protocols and formats can still be operated in parallel with the newly established IT systems (e.g., Data Lake, Advanced Analytics, Cloud).

This way, it is possible to step by step introduce modern communication standards and incre-

mentally migrate to a state-of-the-art Industry 4.0 IT architecture, but still keep the existing IT infrastructure fully operable.



### 1.3.7 Allow Unlimited Scalability

**Rely on unlimited scalability from single equipment and devices to whole facilities.**

SMARTUNIFIER is the first integration platform that allows nearly unlimited virtually scalability in terms of number of connected equipment and devices. The SMARTUNIFIER platform can be applied for integrating one single equipment or device, but with SMARTUNIFIER hundreds or even thousands of equipment and devices within whole facilities can be integrated to upper-level systems

or into the Cloud.

This is because SMARTUNIFIER is not a traditional middleware having a central limiting message bus. Nor does SMARTUNIFIER contain any central performance and latency limiting database for providing its communication features.

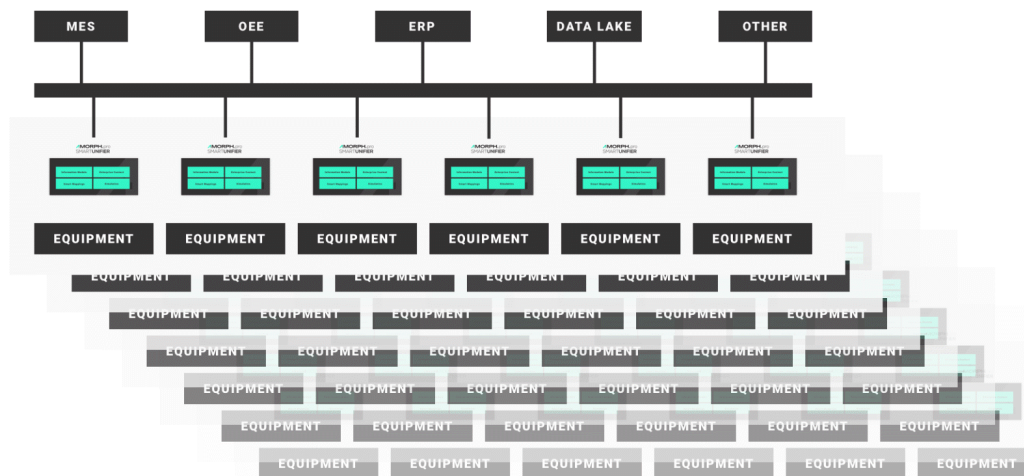
SMARTUNIFIER works as a distributed environment. Using advanced technologies of distributed computing is the key for enormous scalability.

In a large installation a high number of SMARTUNIFIER Instances, each with low software footprint, provide the required communication capabilities. These single instances can be deployed to any location within an enterprise IT network – on a server, on an equipment PC, within the Cloud or on the SMARTUNIFIER Box.

Nevertheless, the configuration of all SMARTUNIFIER Instances can be managed centrally:

- central configuration of Information Models and Smart Mappings
- central Operations Monitoring of installed SMARTUNIFIER Instances.

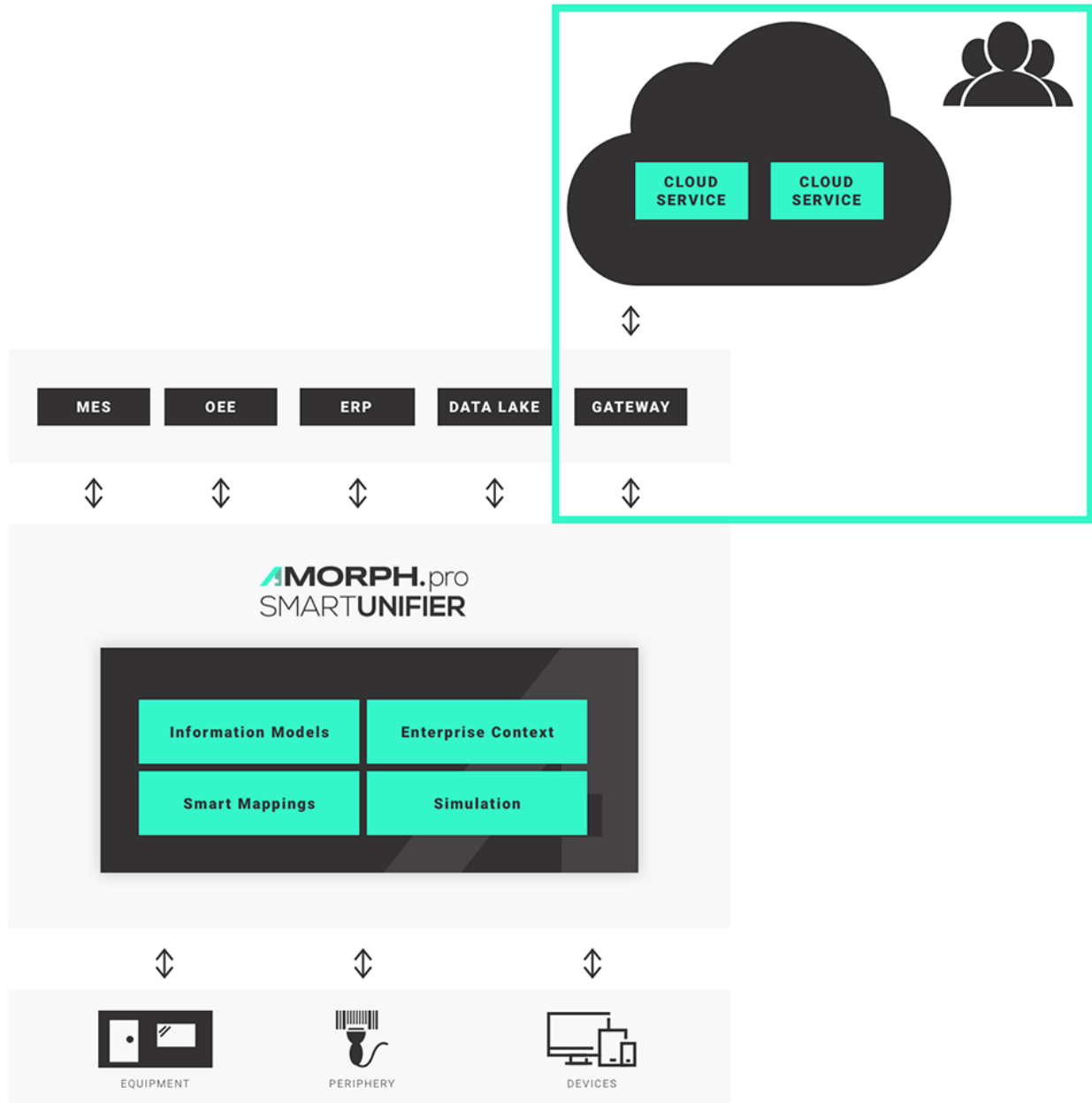
Thus, SMARTUNIFIER is an essential piece of Industry 4.0 for any manufacturing enterprise – allowing fab-wide and enterprise-wide management of production communication and IT integration infrastructure.



### 1.3.8 Enable Internet of Things

**Out-of-the-box connections between equipment, devices and other IT systems to Cloud infrastructures.**

By acting as a translator between equipment and any IOT device precise and secured access of data consumers is possible. The easy connection to any Cloud based infrastructure is also possible (e.g., AWS, Azure).



## 1.4 Connectivity Endpoints and Data Formats

SMARTUNIFIER provides comprehensive connectivity support for a variety of equipment, devices and IT systems. This includes many different pre-configured communication protocols and formats. e.g., OPC-UA, file-based, database, message bus, Webservices and direct PLC connections. Preconfigured interfaces are available also for many standard software applications. A number of these connectivity endpoints / communication protocols require a first time customization from Amorph Systems for a specific customer connectivity use case. Please contact Amorph Systems for detailed information.



### 1.4.1 Connectivity Endpoints / Communication Protocols

The following connectivity endpoints / communication protocols are supported by SMARTUNIFIER.

Table 1: Connectivity Endpoints

Format	Description
ADLink OpenSplice	Connectivity to ADLink OpenSplice middleware via Data Distribution Service (DDS)
AMQP	Interface to AMQP Message Broker via Active MQ
AODB	Interface to various Airport Operational Database (AODB) Systems that support standard communications via e.g., HTTP, REST, SQL
Apache Active MQ	Interface to Active MQ Message Broker
AWS Elastic Container Service (ECS)	Interface to applications running in AWS ECS
AWS Elastic Compute Cloud (EC2)	Interface to applications running in AWS EC2
AWS IoT	Interface to AWS IoT
AWS IoT Greengrass	Interface to AWS IoT Greengrass via MQTT
AWS IoT Sitewise	Interface to AWS IoT SiteWise via OPC-UA
AWS CloudWatch	Interface to CloudWatch
AWS DynamoDB	Interface to AWS DynamoDB
AWS S3	Interface to AWS S3
AWS SNS	Interface to AWS Simple Notification Service (SNS)
AWS SES	Interface to AWS Simple Email Service (SES)
Barcode Reader	Connectivity to any TCP/IP based barcode reader (or other identification system)
Beckhoff	Interface to Beckhoff PLC via Beckhoff OPC-UA Server
DDS	Connectivity to Data Distribution Service (DDS)
File	Read and Write files from arbitrary directories using File Consumer / File Tailer
FTP	Upload and Download files to/from FTP servers
HTTP	Send request to HTTP servers
HTTPS	Send request to HTTPS servers
InfluxDB	Interface to InfluxDB
IBM MQ	Interface to IBM MQ Message Broker
In-Memory	Communication via local machine
JDBC	Access databases through SQL and JDBC (refer to SQL Databases)
JMS	Send and receive messages to/from a JMS Queue or Topic using plain JMS
MES	Interface to a Manufacturing Execution System (MES) that support standard communications via e.g., HTTP, REST, SQL
Modbus-TCP	Communication via Modbus TCP Server / TCP Client
Microsoft Azure (IoT Hub)	Interface to Microsoft Azure Iot Hub via MQTT

Continued on next page

Table 1 – continued from previous page

Format	Description
MTConnect	Communication Interface to MTConnect compliant agent applications
MQTT	Connectivity by implementing MQTT Client
NoSQL Databases	Cassandra, MongoDB, Hbase
OEE	Interface to various Overall Equipment Efficiency (OEE) Applications that support standard communications via e.g., HTTP, REST, SQL
OPC-UA Client	Connectivity by deploying one or multiple OPC-UA Client instances per SMARTUNIFIER Communication Instances
OPC-UA Server	Connectivity by deploying one or multiple OPC-UA Server instances per SMARTUNIFIER Communication Instances
PLC	Connectivity to various PLCs (e.g. Allen-Bradley, B&R, FANUC, General Electric (GE), Hilscher, Honeywell, Krauss Maffei, Mitsubishi, Toshiba, Wago) via TCP/IP
PM	Interface to a various Predictive Maintenance Systems that support standard communications via e.g., HTTP, REST, SQL
REST	Communication via REST using REST Server / REST Client (Web-services)
SAP MII	Interface to SAP MII
SAP RFC	Interface to SAP via remote function call (RFC)
SAP Netweaver	Interface to SAP Netweaver via HTTP
SCADA	Interface to various SCADA Systems that support standard communications via e.g., HTTP, REST, SQL
SECS/GEM	Communication with semiconductor or photovoltaic equipment using SECS/GEM interface protocol for equipment-to-host data communications (TCP/IP).
Siemens Industrial Edge	Deployment of SMARTUNIFIER Communication Instances via Siemens Industrial Edge Platform
Siemens MindSphere (REST)	Interface to MindSphere via REST
Siemens MindSphere (MQTT)	Interface to MindSphere via MQTT
Siemens S7 PLC/TCP	Interface to Siemens S7 1500 / 1200 / 400 / 300 via TCP protocol
Siemens S7 PLC/OPC-UA	Interface to Siemens S7 1500 / 1200 via OPC-UA protocol
Smart Devices	Interface to various Smart Devices (e.g., Smart Phones, Tablets) that support standard communications via e.g., HTTP, REST, SQL
SOAP	Communication via SOAP (Webservices)
Splunk	Interface to Splunk via HTTP Event Collector
Splunk	Interface to Splunk via Metrics Interface
SQL Databases	Interface to any SQL-based database like e.g., DB2, HSQLDB, MariaDB, MSSQL, OracleDB, PostgreSQL, SQLServer and others
TCP	Communication from/to any (binary) TCP based protocol
SFTP	Upload and Download files to/from SFTP servers
UDP	Communication from/to any (binary) UDP based protocol

Continued on next page

Table 1 – continued from previous page

Format	Description
VANTIQ	Interface to VANTIQ
VIPA Speed 7	Interface to VIPA Speed 7 PLC
WAGO PLC/IP	Connectivity to WAGO PLCs via OPC-UA
Websocket	Interface to Websocket Server (TCP/IP)

**Note:** In case a customer requires to connect to other endpoints (e.g., computing devices, PLCs) not listed in the table, please contact Amorph Systems.

## 1.4.2 Data Formats

The following data formats can be used in conjunction with the above defined connectivity endpoints. The possible formats for a certain connectivity endpoint may be restricted based on the selected communication protocol. For detailed information please contact Amorph Systems.

Table 2: Data Formats

Format	Description
Binary	Handling of any binary communication format (e.g., fixed/variable lengths fields, headers/footers)
CSV	Handle CSV (Comma separated values) payloads
JSON	Encode and decode JSON formats
TEXT	Handling of any text-based communication format
XML	Encode and decode XML formats

**Note:** In case a customer requires another data format not listed in the table, please contact Amorph Systems.

## 1.5 What has changed in 1.2.0

**Important: Breaking Change:** This release contains a major update of the SMARTUNIFIER Framework. Instances configured in an older release will not work with this version. Please contact Amorph Systems for guidance on how to migrate SMARTUNIFIER Instances from previous releases.

### Changed

- Improved architecture performance and stability by updating the framework to Scala version 2.13 and Java version 11.

- **Communication Channels:** Improved configuration of Communication Channels by enhancing the internal process of how the configuration forms are generated.
- **Manager UI:** Introduced new icons for several menu entries (Information Model, Mappings, Device Type, Instance, Deployment, Deploy and Undeploy) to improve usability.

**Fixed**

- **Manager UI:** Fixed small UI styling issues.
- **Communication Channel - IsoOnTCPClient:** Fixed configuration issue.
- **Mapping:** Added check to make sure that the Rule name is valid.

## HOW TO INTEGRATE WITH SMARTUNIFIER

Each integration scenario follows the same workflow, which consists out of 5 steps:

1. *Information Models* - describe and visualize communication related data using hierarchical tree structures.
2. *Communication Channels* - describe and configure the protocols needed for the scenario.
3. *Mappings* - define when and how to exchange/transform data between Information Models.
4. *Device Types* - define templates for Instances.
5. *Instances* - define applications that provide the connectivity.

In order to keep artifacts in SMARTUNIFIER organized take a look at:

- Naming Convention for Artifacts
- Group Filter

## 2.1 Information Models

### 2.1.1 What are Information Models

Within the SMARTUNIFIER an Information Model describes the communication related data that is available for a device or IT system. One device or one IT system therefore is represented by one Information Model. An Information Model consists of so-called *Node Types*. Information Models are build up in a hierarchical tree structure, i.e., elements within the Information Model can contain further elements. This is required to model the data structure of devices as naturally as possible.

The kind of *Node Types* to be used depends on the protocol of the device or IT system. Before creating the Information Model take a look in the chapter *Communication Channels* to see which *Node Types* the Channel you want to use is supporting.

### 2.1.2 How to create a new Information Model

Follow the steps described below to create an Information Model:

- Select the SMARTUNIFIER Information Model Perspective (1).



SMARTUnifier Configuration <

Information Models



Channel Types



Communication Channels



Mappings



Device Types



Instances



Deployments



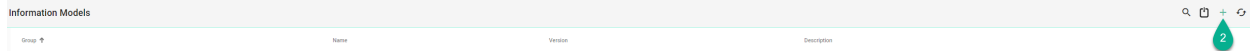
Deployment Endpoints



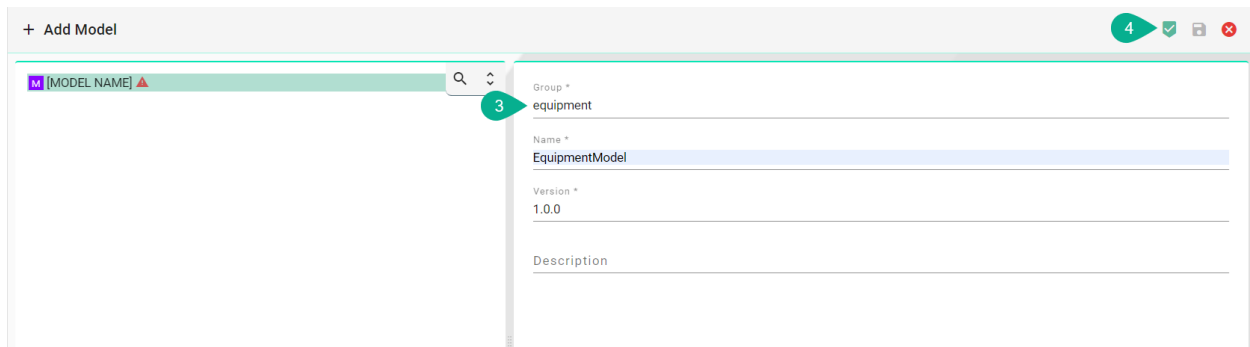
User Management



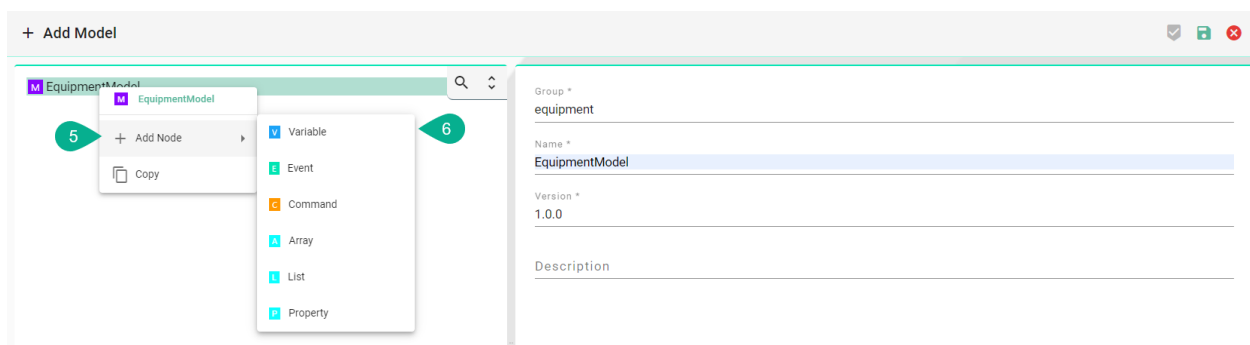
- You are presented with the following screen containing a list view of existing Information Models.
- In order to add a new Information Model, select the “Add Model” button at the top right corner (2).



- On the following screen provide the following mandatory information: Group, Name and Version (3).
- The “Apply” button at the top right corner is enabled after all mandatory fields are filled in. Click the button to generate a new Information Model (4).
- The newly created Information Model is now visible as a node on the left side of the screen.



- After the root model node is created, a new Information Model can be built up using definition types.
- Perform a right click on the root model node and select “Add Node” (5). Select a Definition Type from the dialog (6).



### 2.1.3 Node Types

Model node types are elements within an Information Model. Model node types are variables, properties, events, commands and also collections such as arrays and lists. Each model node type has a Data Type that defines whether the model node type is a predefined data type or a custom data type.

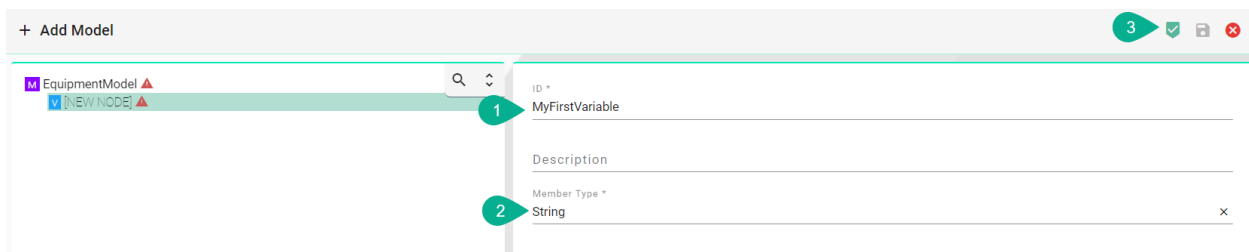
## Variables

### What are Variables

Variables are used to represent values. Within SMARTUNIFIER different types of Variables are defined. They differ in the kind of data that they represent and whether they contain other Variables. For example, a file Object may be defined that contains a stream of bytes. The stream of bytes may be defined as a Data Variable that is an array of bytes. Properties may be used to expose the creation time and owner of the file Object.

### How to create a Variable

- Enter an ID (1)
- Enter a Member Type (2)
- Click the “Apply” button (3)



## Properties

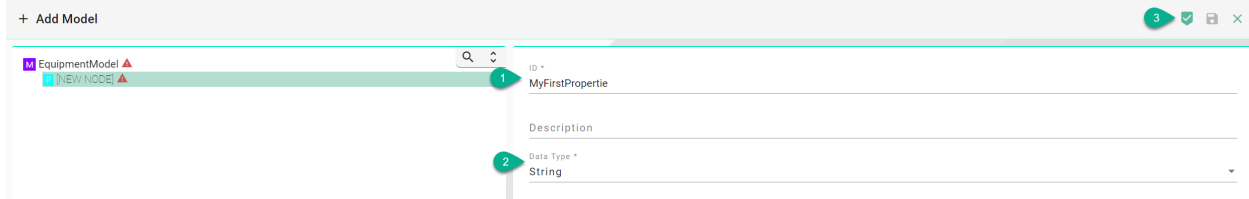
### What are Properties

Properties are working similar to Variables. Properties can be used for XML attributes when XML-files are subject to be processed by SMARTUNIFIER, although XML elements are still represented by Variables in the Information Model.

### How to create a Propertie

- Enter an ID (1)
- Enter a Member Type (2)
- Click the “Apply” button (3)





## Events

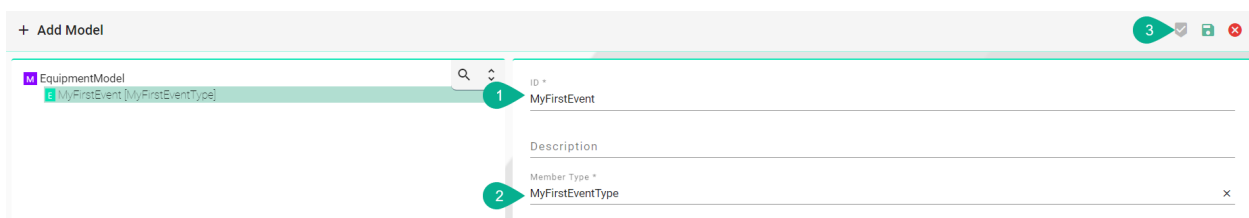
### What are Events

SMARTUNIFIER is an event-driven software. In this context an event is an action or occurrence recognized by SMARTUNIFIER, often originating asynchronously from an external data source (e.g., equipment, device), that may be handled by the SMARTUNIFIER. Computer events can be generated or triggered by external IT systems (e.g., received via a Communication Channel), by the SMARTUNIFIER itself (e.g., timer event) or in other ways (e.g., time triggered event). Typically, events are handled asynchronously with the program flow. The SMARTUNIFIER software can also trigger its own set of events into the event loop, e.g., to communicate the completion of a task. Each event defined in an Information Model has an event type.

An event type consists of one or multiple simple or structured variables. Clients subscribe to such events to receive notifications of event occurrences.

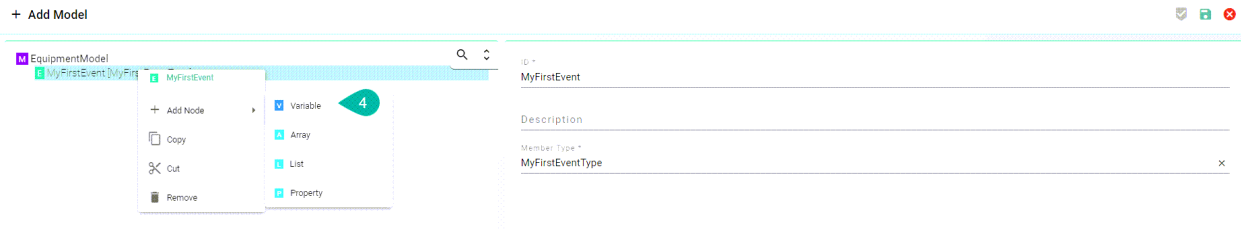
### How to create an Event

- Enter an ID (1)
- Select the Definition Type “Event” from the Drop-Down (2)
- Enter a Member Type for the Event. e.g., “MyFirstEventType” (3)
- Click the “Apply” button (4)



Within the Event Variables, Arrays or Lists can be added. Follow the steps below to add a Variable:

- Right click the Event node, select “Add Node” and choose a Definition Type (4)



- Enter an ID (5)
- Enter a Member Type (6)
- Click the apply button (7)
- Click the “Save” button at the top right corner (8) to save the Information Model



## Commands

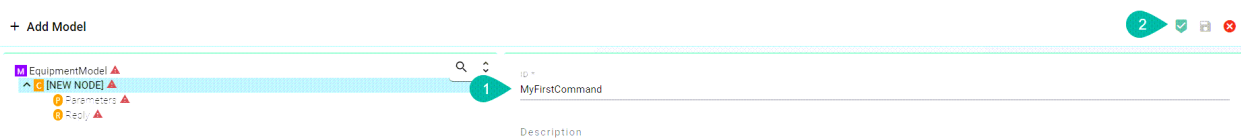
### What are Commands

Commands are functions, whose scope is bound by an owning Information Model, like the methods of a class in object-oriented programming. Commands within an Information Model are typically invoked by an external IT system (e.g., an equipment) that triggers the command. In addition, commands of a target Information Model (e.g., an MES) can be triggered by the SMARTUNIFIER through a Mapping. A command contains one or multiple simple or structured Variables. Also a command has a return parameter that likewise can be a simple or complex data type.

The lifetime of the command invocation instance begins when the client calls the command and ends when the result is returned. While commands may affect the state of the owning model, they have no explicit state of their own. In this sense, they are stateless. Each command defined in an Information Model has a command type

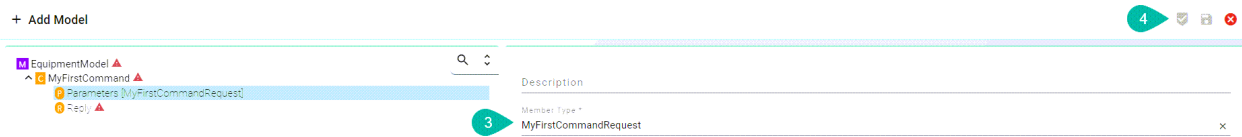
### How to create a Command

- Enter an ID (1)
- Click the “Apply” button (2)

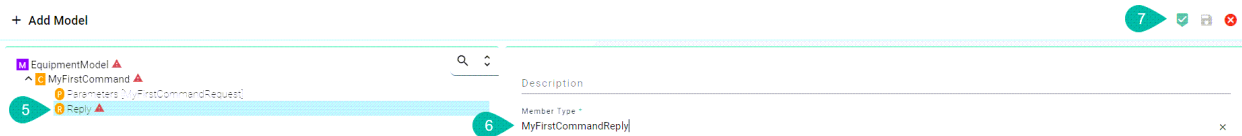


The main two parts of a Command are the Request, referred to as Parameters within the SMARTUNIFIER, and the Reply. Variables, Arrays and Lists can be added to both of these command parts. Follow the steps below to add a Variable to Parameters :

- Select the Parameters node from the tree
- Enter a Member Type (3)
- Click the “Apply” button (4)

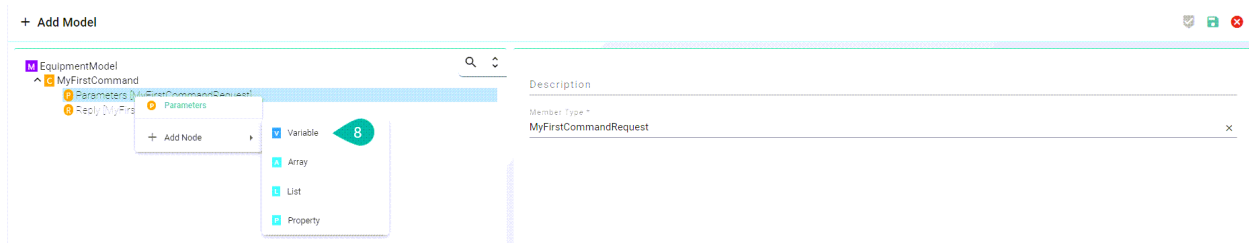


- Select the Reply node from the tree (5)
- Enter a Member Type (6)
- Click the “Apply” button (7)



Follow the steps below to add nodes under the Parameter and Reply node:

- Right click the Parameter node, select “Add Node” and choose a Definition Type (8)



- Enter an ID (9)
- Enter a Member Type (10)
- Click the “Apply” button (11)
- Click the “Save” button (12) to save the Information Model



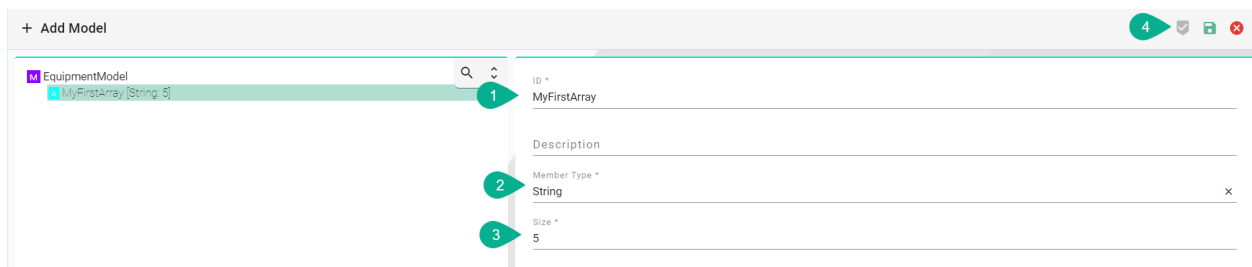
## Arrays

### What are Arrays

Arrays allow to hold a collection of elements that have the same type.

### How to create an Array

- Enter an ID (1)
- Select a Member Type for the Array by clicking the Member Type Drop-Down (2)
- Enter the size of the Array (3)
- Click the “Apply” button (4)



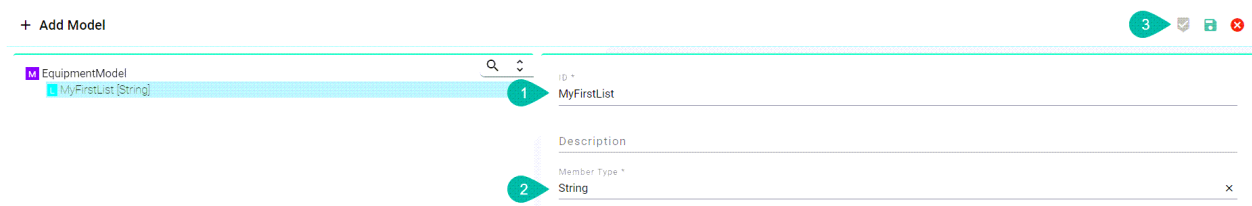
## Lists

### What are Lists

Lists allow to hold a collection of elements (*Variables*).

### How to create a List

- Enter an ID (1)
- Enter a Member Type for the List. E.g., “String” (2)
- Click the “Apply” button (3)



### 2.1.4 Data Types

There are two kinds of Data Types:

- Predefined Types e.g., String, Integer, Boolean and more. (Note: Only available for the definition types - Variables, Properties, Arrays, Lists)
- Custom Types

#### How to create a Variable as a Simple Type

- Add a new Variable, enter an ID and select a primary data type for the Data Type e.g., “String” (1)

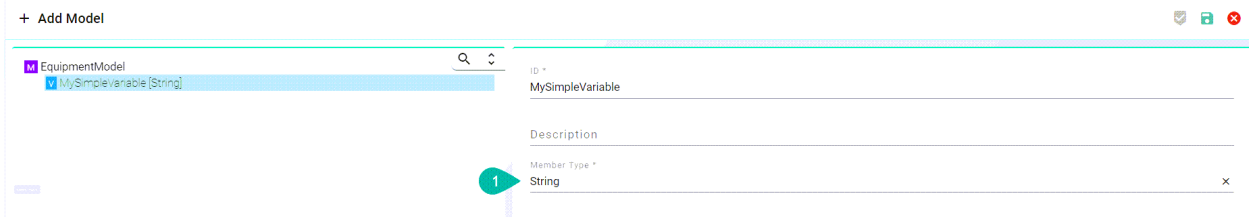


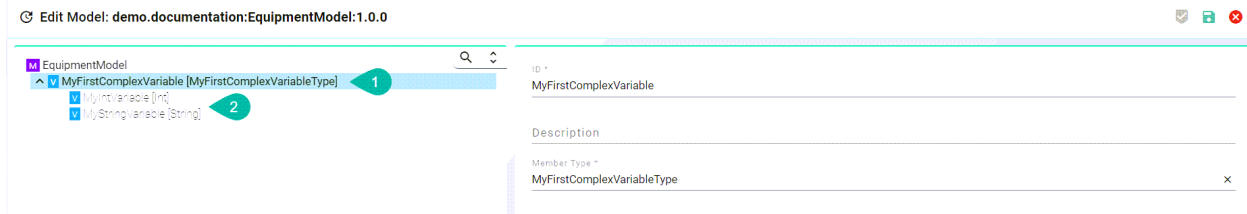
Table 1: Predefined Data Types

Type	Definition
Boolean	true or false
Byte	8 bit signed value (-27 to 27-1)
Int	32 bit signed value(-231 to 231-1)
String	Sequence of characters
Char	16 bit unsigned Unicode character(0 to 216-1)
Double	64 bit IEEE 754 double-precision float
Float	32 bit IEEE 754 single-precision float
Long	64 bit signed value(-263 to 263-1)
Short	16-bit signed integer
Array	Mutable, indexed collections of values.
List	Class for immutable linked lists representing ordered collections of elements.
LocalDate	Immutable date-time object that represents a date, often viewed as year-month-day.
LocalDateTime	Immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second.
LocalTime	Immutable date-time object that represents a time, often viewed as hour-minute-second.
OffsetDateTime	Immutable representation of a date-time with an offset.

## How to create a Variable as a Custom Type

- Add a new Variable, enter an ID and enter a custom name for the Data Type e.g., “MyFirstComplexVariableType” (1)
- Select the Custom Variable - “MyFirstComplexVariableType” - and add a new Variable underneath it (2)

**Note:** Model Node Types with custom data types can be easily duplicated throughout the Information Model by selecting the same custom data type for a new model node type.



Data Types for Properties, Arrays and Lists can be defined as shown above for Variables.

## 2.2 Communication Channels

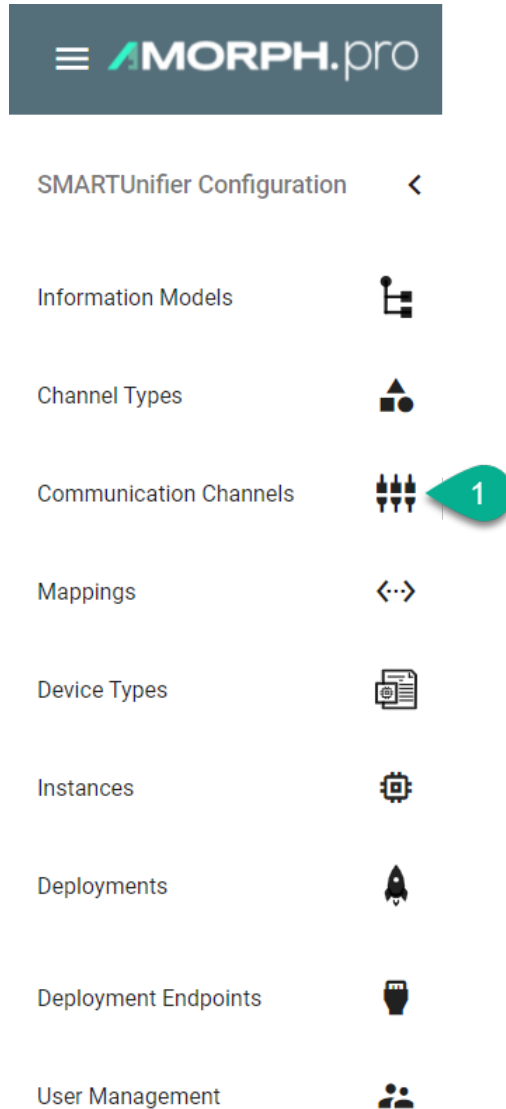
### 2.2.1 What are Channels

Communication Channel or simply Channel refers to a transmission medium. A Channel is used to convey information from one or several senders (or transmitters). Communicating data from one location to another requires a pathway or medium. These pathways are called Communication Channels, and the information is transmitted with the help of communication protocols. Each Information Model can have one Channel or many, and each model can choose which Channels it subscribes to. The information is transmitted through the Communication Channels in both directions: from the external system to the SMARTUNIFIER application and vice versa.

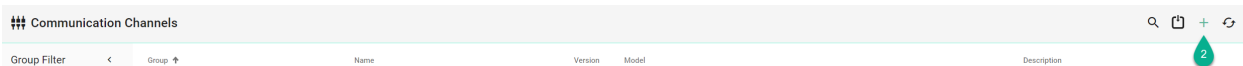
### 2.2.2 How to create a new Channel

Follow the steps below to create a new Channel:

- Go to the Communication Channels perspective by clicking the “Communication Channels” button (1)



- To create a new Channel, select the “Add Channel” button at the top right corner (2)



- The creation of a Communication Channel is split up into two parts. First enter basic information about the new Communication Channel
  - Fill in the information for the Channel identifier such as: Group, Name and Version. Description is optional (3)
  - Besides that, associate the Channel with an Information Model (4)
  - Select the type this Channel represents from the Drop-Down (5). A list of available Channel Types and a description of how to configure each of them can be found below
- Click the “Save” button (6) to save the Channel

③ Add Communication Channel

6

3 Group \*  
channel.demo.enterprise

Name \*  
EnterpriseChannel

Version \*  
1.0.0

Description

4 Model  
demo.productionmonitoring.unifier.EnterpriseModel.1.0.0

5 Channel type \*  
REST Server

### 2.2.3 Channel Types and Configuration

There are several Channel Types available with SMARTUNIFIER. The supported Communication Channel Types are listed in the chapter *Connectivity Endpoints / Communication Protocols*. If a specific Communicating Channel Type is not available in this product version, please contact Amorph Systems. In many cases the provision of a specific Communication Channel Type can be provided as extension to the standard product.

The configuration of the Communication Channels can be done on Channel, Device Type and Instance level.

---

**Note:** Important to note is that the configuration of a Channel can be overwritten accordingly. For example: The configuration done in the Communication Channel view can be changed in the Device Type or Instance view.

---

The following paragraphs lay out the configuration process of selected Channel Types. If the Channel Type you want to use is not described, please contact Amorph Systems for configuration guidance.

#### File-based

##### File Tailer

##### Characteristics:

- File Tailer monitors a given file in a given location.
- Data is processed line by line.
- Note that the File Tailer does not support the definition type **List** in the Information Model.

##### Supported File Formats:

- CSV
- JSON
- XML

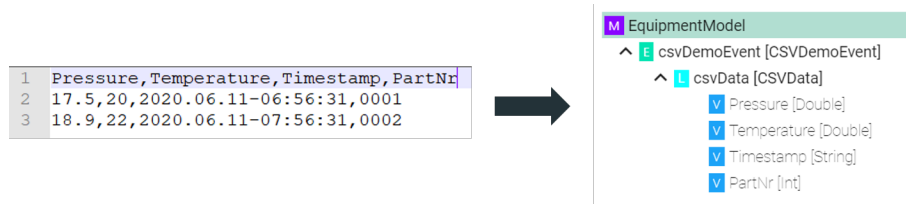


## Information Model Requirements

The first Node after the root node **M** must be of type Event **E**.

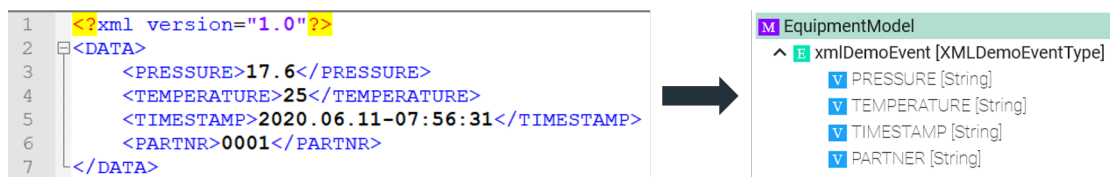
### CSV

- The node after the Event must be of type List **L** - multiple lines, each representing a data record.
- Fields, which are separated by commas, are represented by the Node Type Variable **V**. Note that the order of fields in the CSV file must match the order of Variables in the Information Model.



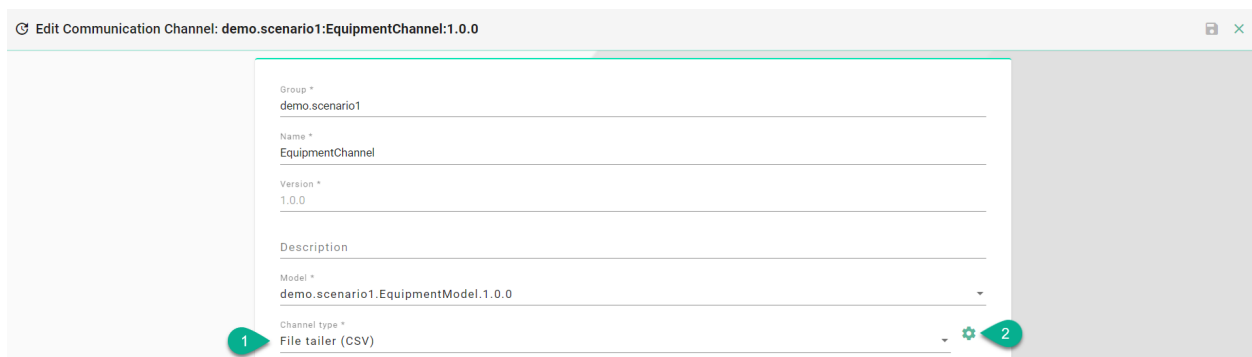
### XML

- Elements of the XML file are represented by the Node Type Variable **V**.
- Attributes of the XML file are represented by the Node Type Property **P**. In order to assign attributes to elements in the Information Model, the element Node Type **V** must be a *Custom Data Type*.



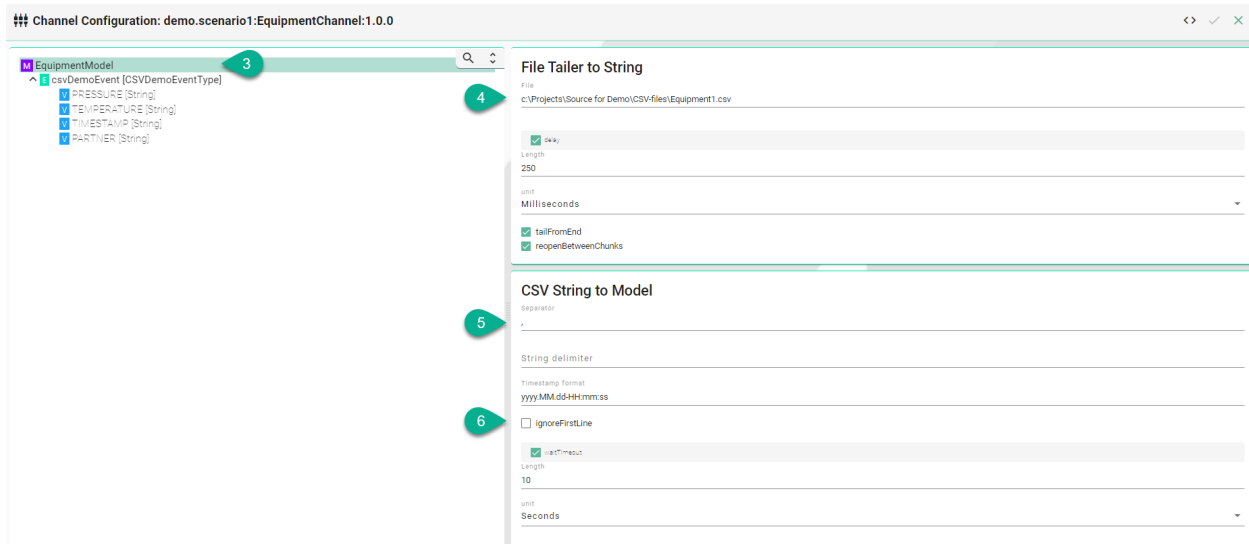
## How to use File Tailer with CSV

1. Select **File tailer (CSV)** from the Drop-Down.
2. Click the **Configure** button.



3. **Make sure** the root model node is selected to be able to configure the File Tailer to String and CSV String to Model.

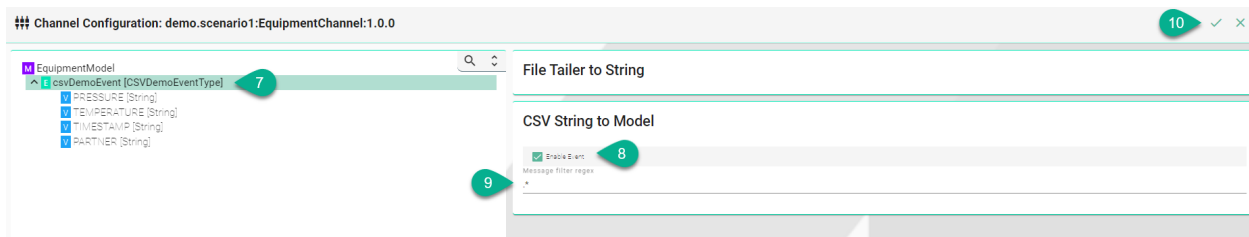
4. Enter the **file path** for the CSV-file on your machine.
5. Enter the **separator** which is used in the CSV-file, the **string delimiter** and the **timestamp format** if one is used.
6. If the CSV file contains a header enable **ignoreFirstLine**.



7. Select the **event node** in the tree on the left side.

**Note:** The entries of a CSV-File can only be mapped directly to an Event object and its parameters.

8. Check the **routes** checkbox.
9. Enter a **regular expression** for the message filter.
10. Click the **Apply** button, then the **Close** button and save the Channel by clicking the **Save** button on the upper right corner.



**Description of configuration properties:**

Property	Description	Example
Separator	Separator type, used in the csv file	, ;
Delimiter	Values that have an additional delimiter like “Date”, “Time”	”
Timestamp format	Format of the timestamp	YYYY-MM-DD HH:mm:ss
File	Path to the csv file	C:\test.csv
Delay Millis	Delay between checks of the file for new content in milliseconds	250
TailFromEnd	Set to true to tail from the end of the file, false to tail from the beginning of the file	true, false
ReopenBetweenChunks	If true, close and reopen the file between reading chunks	true, false
routes	Path of a node in the Information Model	true, false
messageFilter-Regex	Regular Expression for the message filter used in the implementation	.*

## File Consumer

### Characteristics

- File Consumer monitors a specified folder - the so-called input folder
- If a file is inserted the following actions take place:
  - The Trigger of the specified Rule in the Mapping is activated
  - Thus, the Rule is executed
- After successful execution of the rule the file is moved into a so-called output folder
- In case of an exception the file is moved into an error folder

### Supported File Formats:

- CSV
- JSON
- XML

### Information Model Requirements

The first Node after the root node **M** must be of type Event **E**.

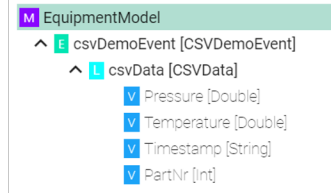
### CSV

- The node after the Event must be of type List **L** - multiple lines, each representing a data record.
- Fields, which are separated by commas, are represented by the Node Type Variable **V**. Note that the order of fields in the CSV file must match the order of Variables in the Information Model.

```

1 Pressure, Temperature, Timestamp, PartNr
2 17.5, 20, 2020.06.11-06:56:31, 0001
3 18.9, 22, 2020.06.11-07:56:31, 0002

```



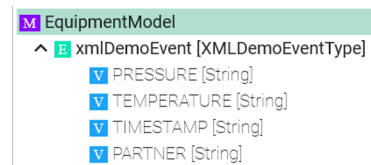
## XML

- Elements of the XML file are represented by the Node Type Variable **V**.
- Attributes of the XML file are represented by the Node Type Property **P**. In order to assign attributes to elements in the Information Model, the element Node Type **V** must be a *Custom Data Type*.

```

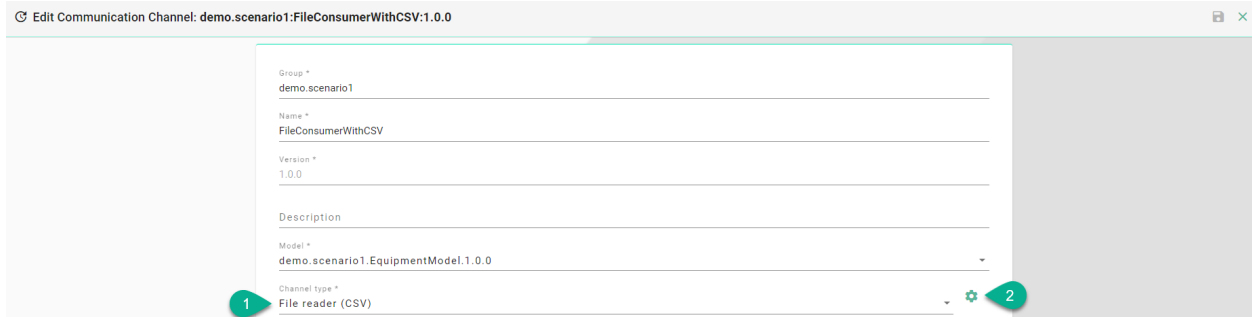
1 <?xml version="1.0"?>
2 <DATA>
3   <PRESSURE>17.6</PRESSURE>
4   <TEMPERATURE>25</TEMPERATURE>
5   <TIMESTAMP>2020.06.11-07:56:31</TIMESTAMP>
6   <PARTNR>0001</PARTNR>
7 </DATA>

```



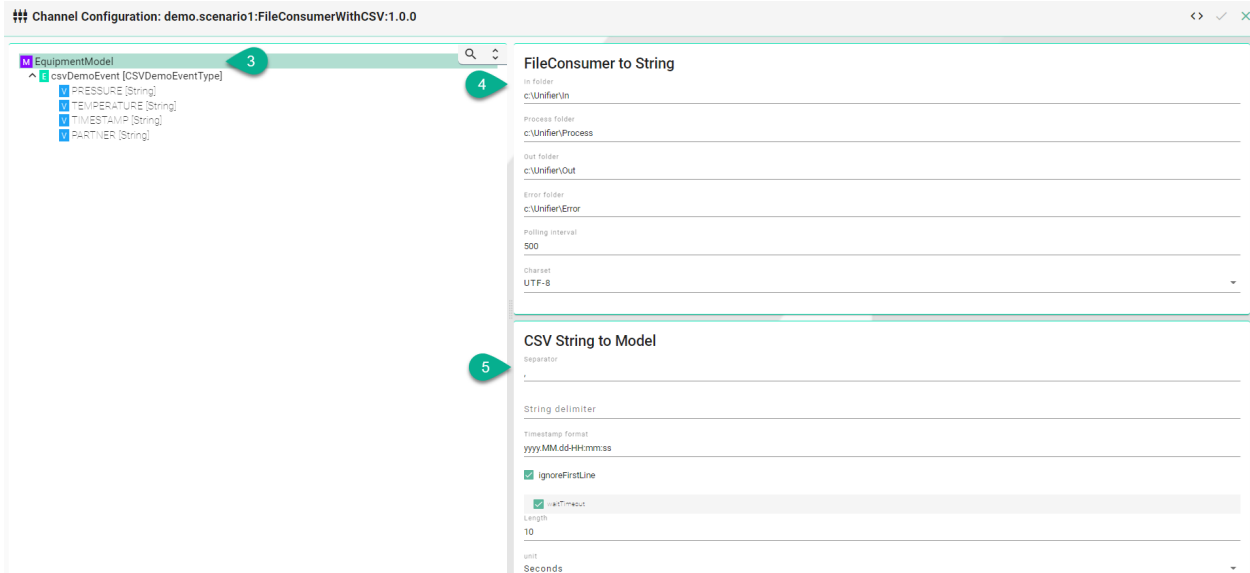
## How to use File Consumer with CSV

1. Select **File reader (CSV)** from the Drop-Down.
2. Click the **Configure** button.



3. **Make sure** the root model node is selected to configure the File Consumer to String as well as the CSV String to Model.
4. File Consumer to String - Configuration
  - Enter a path for the input folder - **InFolder**
  - Enter a path for the process folder - **ProcessFolder**
  - Enter a path for the output folder - **OutFolder**
  - Enter a path for the error folder - **ErrorFolder**
  - Select the **CharSet** according to the file in use
5. CSV String to Model - Configuration

- Enter the **separator** which is used in the CSV-file
- If needed: Set **string delimiter** and/or the **timestamp format**
- If the CSV file contains a header enable **ignoreFirstLine**



6. Specify the Event used by selecting the **event node** in the tree on the left side

**Note:** The entries of a CSV-File can only be mapped directly to an Event object and its parameters.

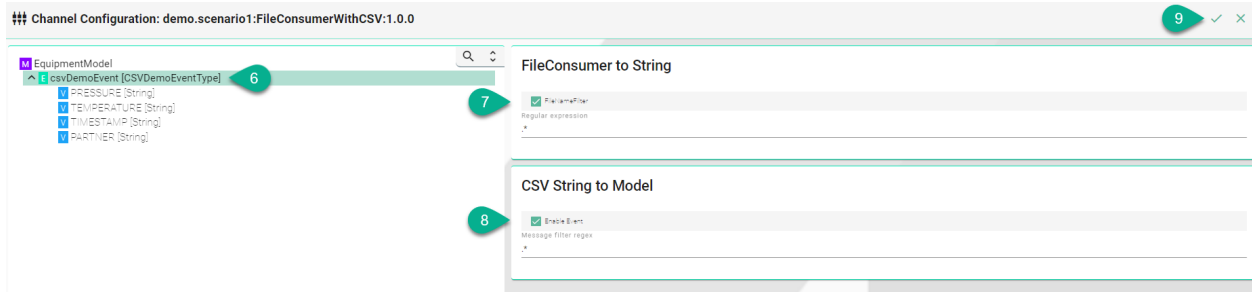
#### 7. File Consumer to String - Configuration

- Enable the **FileNameFilter** checkbox
- Enter a **regular expression** in order to determine which file is to be processed in the input folder

#### 8. Csv String to Model - Configuration

- Enable the **routes** checkbox
- Start of processing
  - If the entire content of the file is processed on this event enter a wildcard in the **RegEx** field
  - If the processing starts at a specific line enter a regular expression in the **RegEx** field in order to identify the line

9. Click the **Apply** button, then the **Close** button and save the Channel by clicking the **Save** button



**Description of configuration properties:**

Property	Description	Example
Separator	Separator type, used in the csv file	, ; ;
Delimiter	Values that have an additional delimiter like “Date”, “Time”	”
Timestamp format	Format of the timestamp	YYYY-MM-DD HH:mm:ss
ignoreFirst-Line	Delay between checks of the file for new content in milliseconds	true, false
TailFromEnd	Set to true to tail from the end of the file, false to tail from the beginning of the file	true, false
InFolder	Path leading to the Input Folder	C:\FileConsumer\In
OutFolder	Path of a node in the Information Model	C:\FileConsumer\Out
ErrorFolder	Regular Expression for the message filter used in the implementation	C:\FileConsumer\Error
CharSet	Encoding of the file in use	UTF-8, `UTF-8 BOM, ..
Process-Folder	Regular Expression for the message filter used in the implementation	C:\FileConsumer\Process

**Databases**

**InfluxDB**

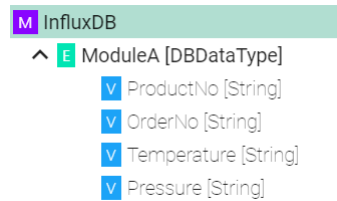
**Characteristics**

In case of a time series data use case where you need to ingest data in a fast and efficient way you can use [InfluxDB](#).

**Information Model Requirements**

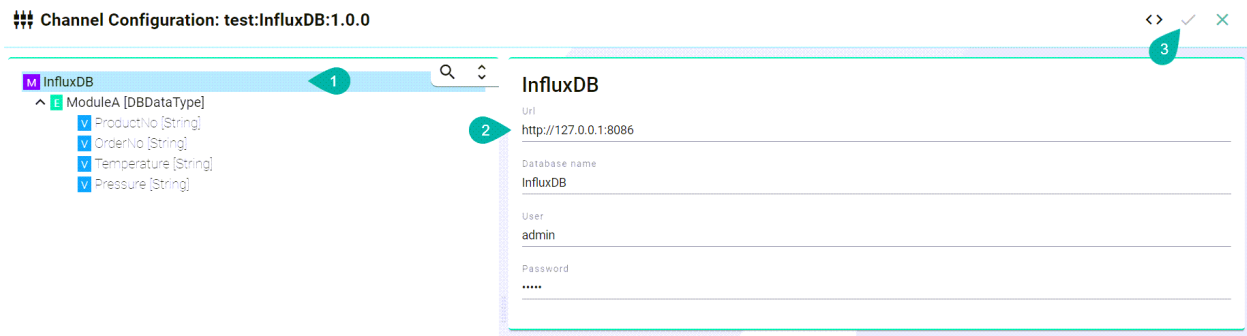
**Inserts**

- The node after the root model node must be of type Event **E** which represent a database table.
- Columns of databases are represented by Variables **V**.



## How to configure InfluxDB

1. Select the **root model node** in the tree on the left.
2. Configure the InfluxDB
  - Enter the **URL** to the database
  - Enter the **database name**
  - Enter the database **user name** and the **password**
3. Click the **Apply** button and save the Channel by clicking the **Save** button on the upper right corner.



## SQL Database

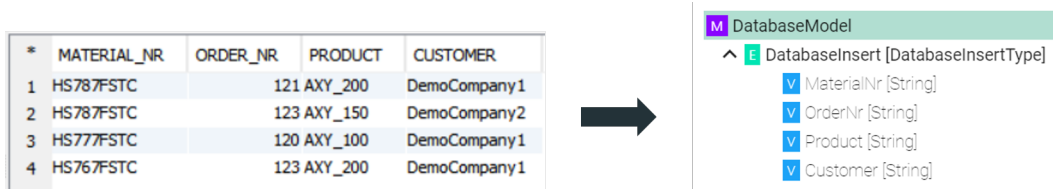
### Characteristics

- The SQL Channel can be configured for the following two scenarios:
  - Inserting data
  - Updating data
  - Retrieving data
- When inserting values into the database please **note** that “infinity” values are converted automatically into “null” values.

### Information Model Requirements

#### Insert/Update

- The node after the root model node must be of type Event **E** which represent a database table.
- In case of relational databases: Tables which are dependent on each other require a List **L**.
- Columns of databases are represented by Variables **V**.



### Select

- The Command **C** defines that after a request is made, a reply with a result is expected.
- Parameters **P** within a Command represent a collection of query parameter - query parameters are defined as Variables **V**.
- Reply **R** within a Command represents the result of the Command - results are defined as Variables **V**.



### How to configure the SQL-Database

1. Select the **root model node** in the tree on the left.
2. Configure the database connection
  - Select the **database type**.
  - Specify a **reconnection interval**.
  - Enter the **database connection url** for the specific database type.
    - **DB2**: jdbc:db2:server:port/database
    - **HSQldb**: jdbc:hsqldb:file:databaseFileName;properties
    - **ORACLE**: jdbc:oracle:thin:prodHost:port:sid
    - **PostgreSQL**: jdbc:postgresql://host:port/database
    - **SQLServer**: jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]
    - **MariaDB**: jdbc:(mysql|mariadb):[replication:|loadbalance:|sequential:|aurora:]/<host>[:<portnumber>]/[database][?<key1>=<value1>[&<key2>=<value2>]]



- Enter the database user **name** and **password**.

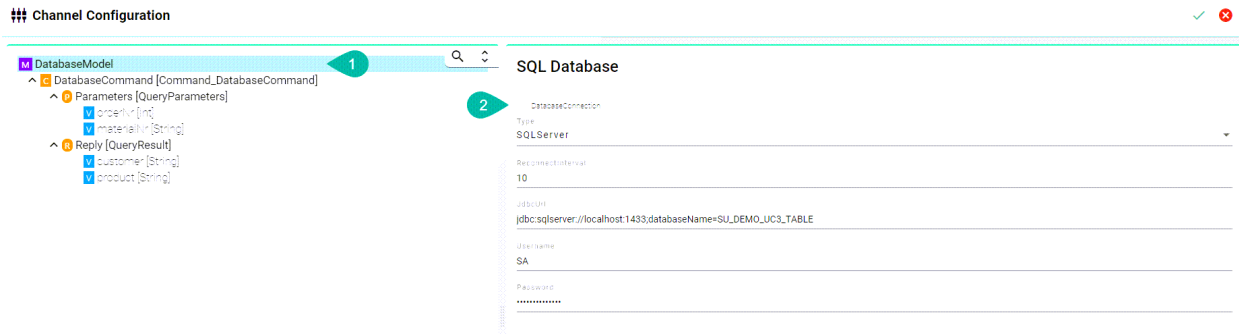


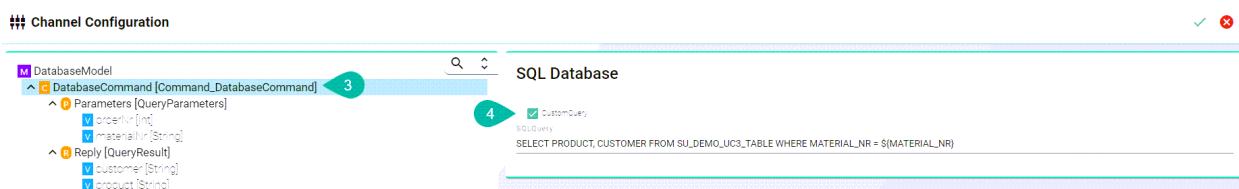
Table 2: Description of database configuration properties

Property	Description	Ma	Example
<b>Type</b>	Type of the database	Yes	MariaDB, SQLServer, ORACLE, HSQLDB, DB2, PostgreSQL
<b>ReconnectInterval</b>	Time to reconnect if connection fails	Yes	10 (in milliseconds)
<b>JdbcUrl</b>	Url to connect to database	Yes	SQLServer: jdbc:sqlserver://<serverName:1433; databaseName=unifier MariaDB: jdbc:mariadb://localhost:3306/unifier? connectTimeout=5000 DB2: jdbc:db2:/ /127.0.0.1:50000/TESTDB HSQLDB: jdbc:hsqldb:file:\$dbFileName;shutdown=true ORACLE: jdbc:oracle:thin:@localhost:1521/ MYCDB PostgreSQL: jdbc:postgresql://127.0.0.1:5432/postgres
<b>Username and password</b>	Credentials of the database	Yes	

**Note:** The configuration of specific information model nodes differs whether you want to perform an **insert** or an **select** statement on the database. Inserting data into the database requires an **event node** whereas selecting data requires a **command node** in the Information Model.

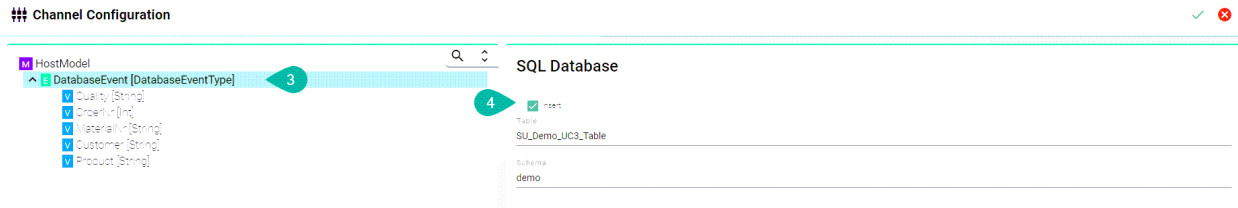
### Select Statement

3. Select the **command node** in the tree on the left.
4. Check the **custom query** checkbox and enter the **sql query**.

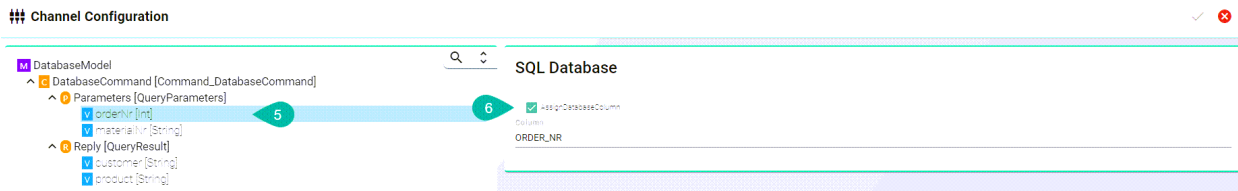


## Insert Statement

3. Select the **event node** in the tree on the left.
4. Check the **insert** checkbox and enter the **table name**. If required enter a **schema name**.



5. Each variable under *Parameters* and *Reply* needs to be assigned to a database column. Select the **variable node** under *Parameters* and in the tree select what needs to be configured.
6. Check the **assign database column** checkbox and enter the **column name** as it is defined in the used database.



## Protocols

### MQTT

### Characteristics

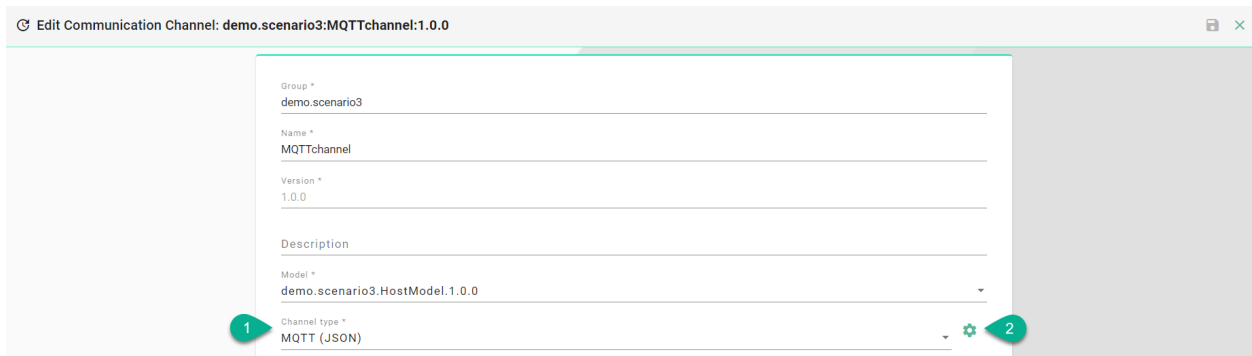
### Information Model Requirements

- The first Node after the root node **M** must be of type Event **E**.
- The following Node Types can be used under the Event Node:
  - Variables **V** with a *Simple Data Type* represents the key-value pairs.
  - Variables **V** with a *Custom Data Type* represent objects that can contain key-value pairs.
  - With Lists **L** you can aggregate multiple variables.
- In case of publishing a topic, the Information Model determines the structure of the payload.
- In case of subscribing to a topic make sure that the Information Model structure matches the payload.

### How to configure the MQTT Channel

1. Select the **MQTT (JSON)** as Channel Type.

## 2. Click the **Configure** button.

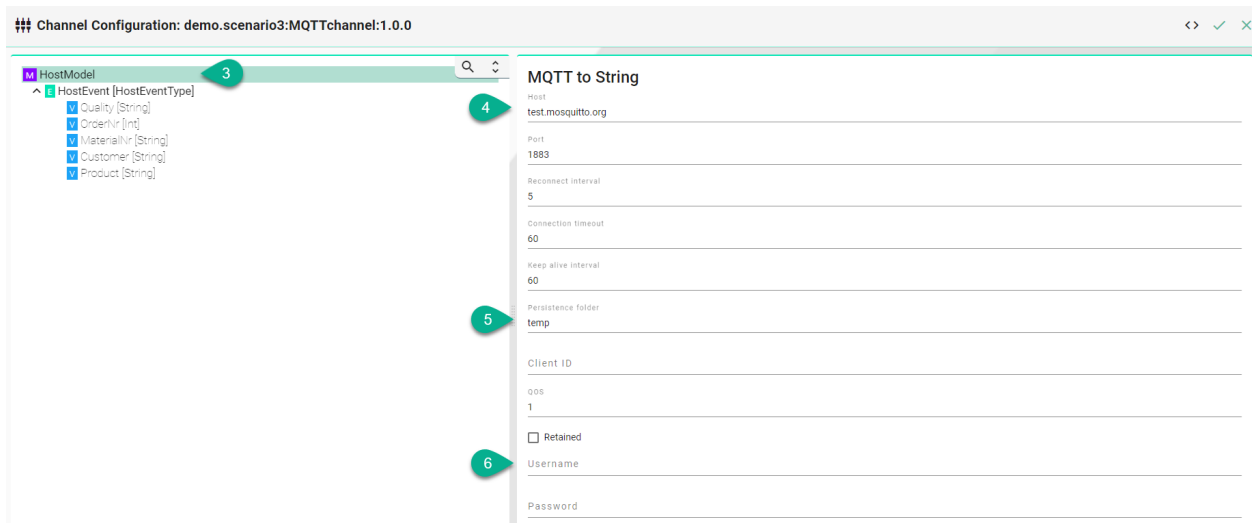


## 3. Select the **root model node**

## 4. Enter **host** and **port** of the MQTT Broker used.

## 5. Specify a path to a folder on your local machine. The **temp** directory inside the SMARTUNIFIER Manager can be also used.

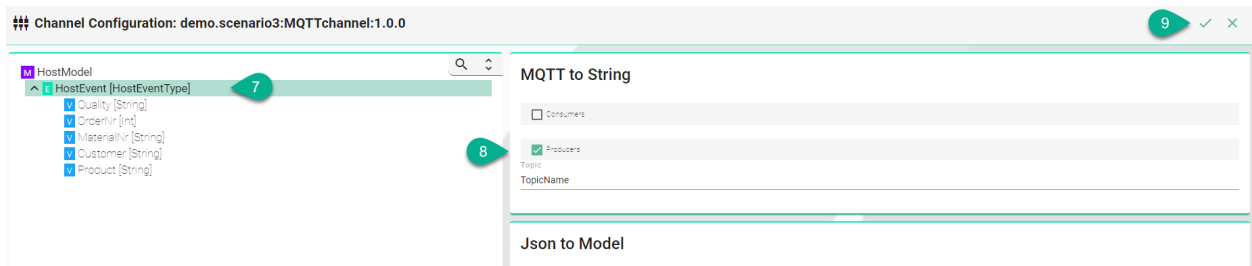
## 6. Enter **user name** and **password**. If there are no credentials needed (e.g., test.mosquitto.org) **make sure** the fields remain empty.



## 7. Select the **event node** in the tree on the left.

## 8. Enable either **producer** or **consumer** depending on the use case and enter a **topic name**.

## 9. Click the **Apply** button.



## Certificates

Encrypted connection using TLS security is supported. Follow the steps below to encrypt the connection.

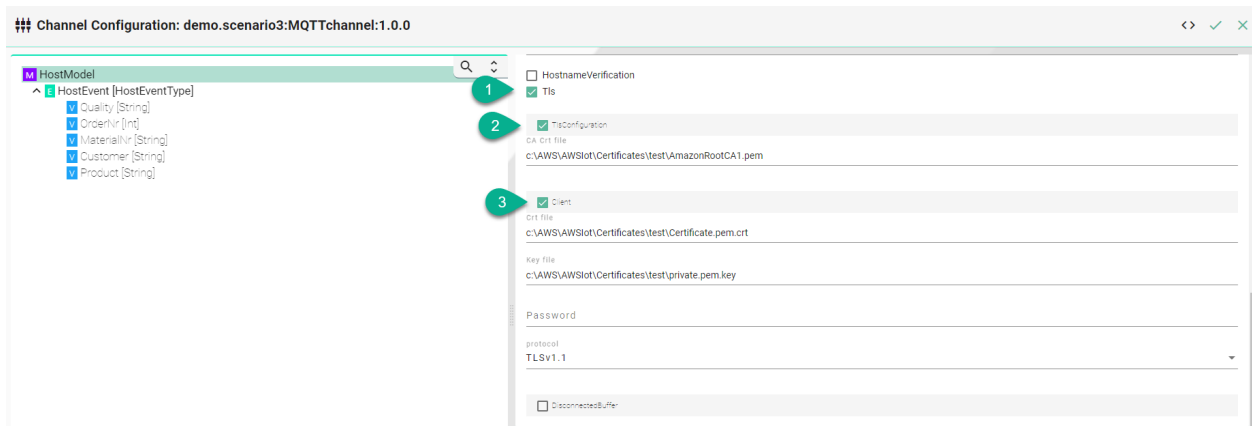
1. Enable the **Tls** checkbox
2. Enable the **Tls Configuration** checkbox
  - Enter the **path** to the **CA (certificate authority) certificate** of the CA that has signed the server certificate

---

**Note:** Make sure the CA certificate is valid.

---

3. Enable the **Client** checkbox
  - Enter the **path** to the **client certificate**. The client certificate identifies the client just like the server certificate identifies the server.
  - Enter the **path** to the **private client key**.
  - If applicable enter the **password**
  - Select the **protocol** from the Drop-Down.



## Disconnected Buffer

In case the connection is lost, messages can be buffered offline when the Disconnected Buffer is enabled. Follow the steps below to enable the DisconnectedBuffer.

1. Enable the **DisconnectedBuffer** checkbox.
2. Set the **Buffer Size** - defines the amount of messages being hold e.g., 5000.
3. (Optional) Enable **PersistBuffer**.
4. (Optional) Enable **DeleteOldestMessage**.

**Description of configuration properties:**



Property	Description	Example
host	URL of the MQTT Broker.	test. mosquitto.org
port	Port of the MQTT Broker.	1883
reconnectInterval	Time interval to reconnect to the MQTT Broker after loss of connection in seconds	5
connection-Timeout	Time interval the connection times out in seconds	60
keepAliveInterval	Time the session persists in seconds	60
persistence-Folder	Path to a folder for the persistence store of the MQTT	temp
clientId	Identifies an MQTT client which connects to an MQTT Broker	MyClientID
username	Client username	Username
password	Client password	Password
hostnameVerification	Hostname Verification	true, false
tls	Encryption	true, false
producers	Data producer	true, false
consumer	Data consumer	true, false
protocol	TLS protocol version	TLSv1.1, TLSv1.2
disconnected-Buffer	Offline buffering of data	true, false
bufferSize	Amount of message allowed in the buffer	5000
persistBuffer	Buffer persistence	true, false
deleteOldestMessage	Delete oldest message in buffer	true, false

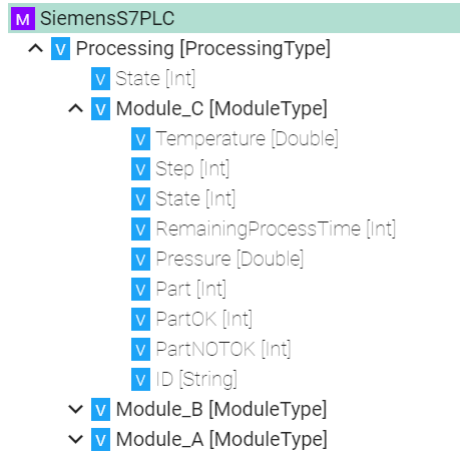
## OPC-UA

### Characteristics

OPC (Open Platform Communications) enables access to machines, devices and other systems in a standardized way. To learn more about the standard visit the [OPC-UA website](#).

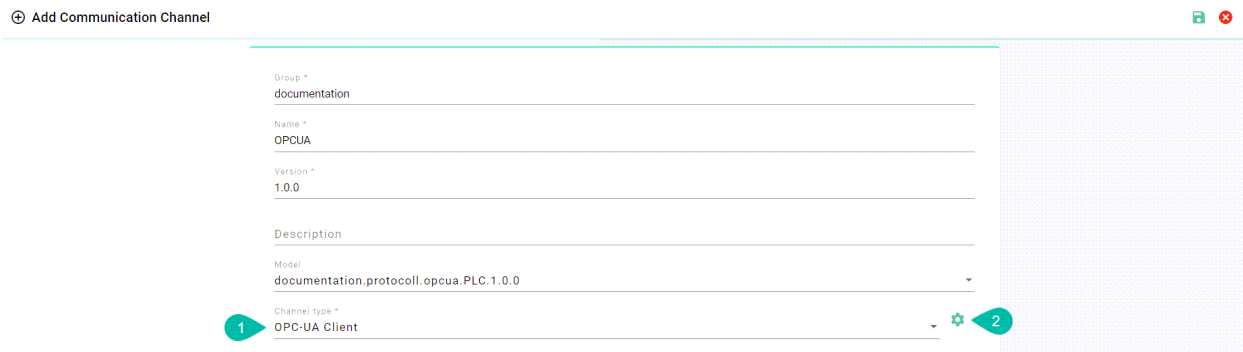
### Information Model Requirements

- The following Node Types can be used to model data structures:
  - Variables  with a *Simple Data Type*.
  - Variables  with a *Custom Data Type*.



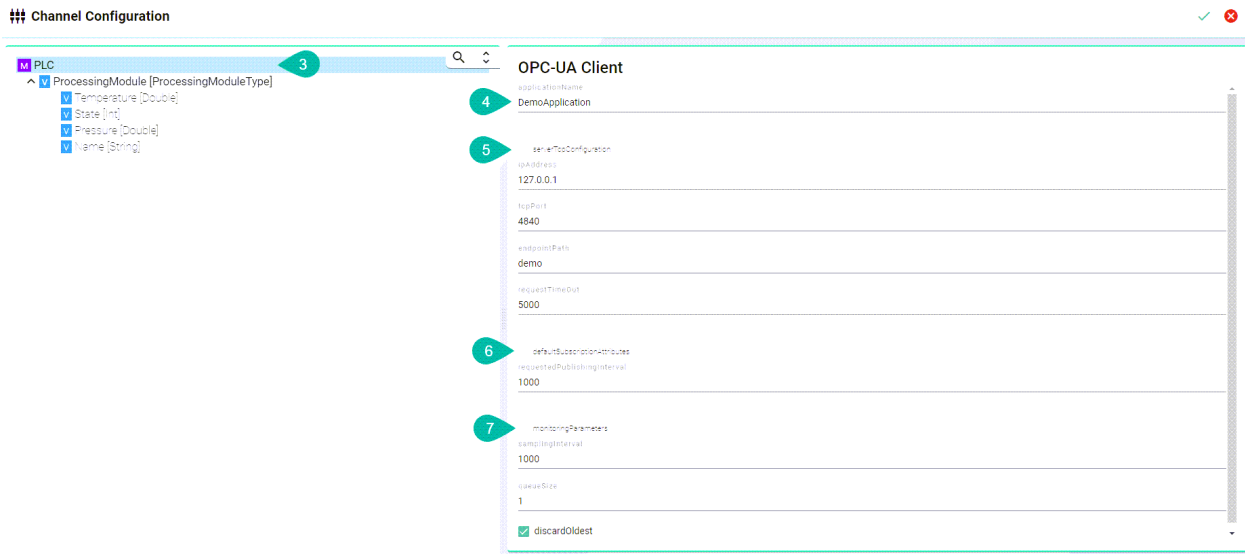
## OPC-UA Client

1. Select **OPC-UA Client** as Channel Type.
2. Click the **Configure** button.

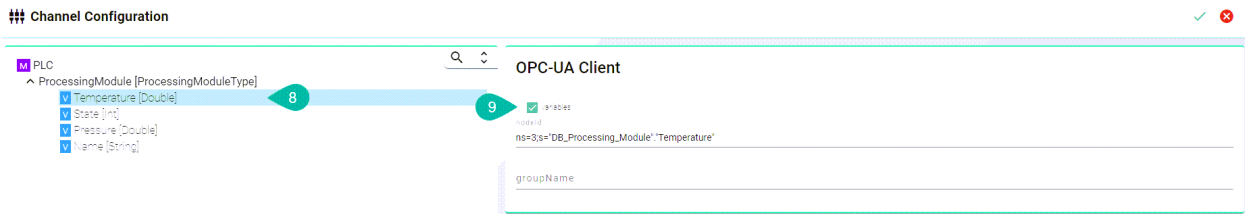


3. **Make sure** the root model node is selected to configure the OPC-UA Client
4. Enter an **applicationName**
5. Configure the serverTcpConfiguration
  - Enter an **ipAdress**
  - Enter the **port**
  - Define an **endpoint**
  - Set a **requestTimeOut**
6. Configure the defaultSubscriptionAttribute
  - Define a **publishingInterval**
7. Configure monitoringParameters
  - Set a **samplingInterval**

- Enter a **queueSize**
- Enable **discardOldest** depending on the use case



8. Assign OPC-UA data block variables to corresponding variables in the Information Model by selecting the variable in the tree
9. Assign data block
  - Enable the **variables** checkbox
  - Enter the **nodeId**

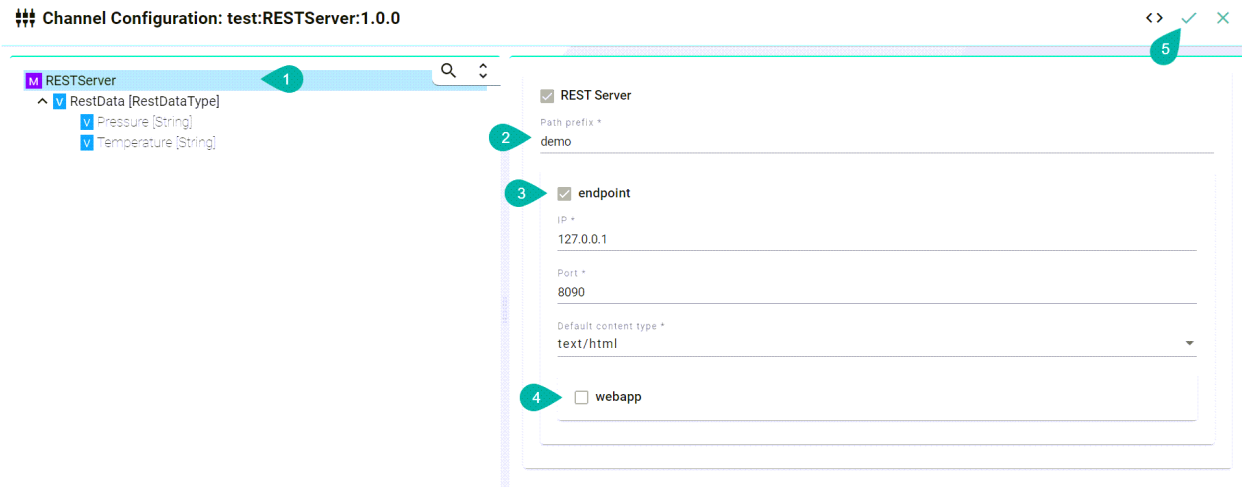


## REST

### REST Server

1. Select the **root model node** in the tree on the left.
2. Enter a **path prefix**.
3. Configure the REST Server endpoint.
  - Enter the **IP**.
  - Enter the **port**.
  - Enter the **Content-Type**.

4. Check the **webapp** checkbox and provide the **WAR-file** if you want to host an application.
5. Click the **Apply** button and save the Channel by clicking the **Save** button on the upper right corner.



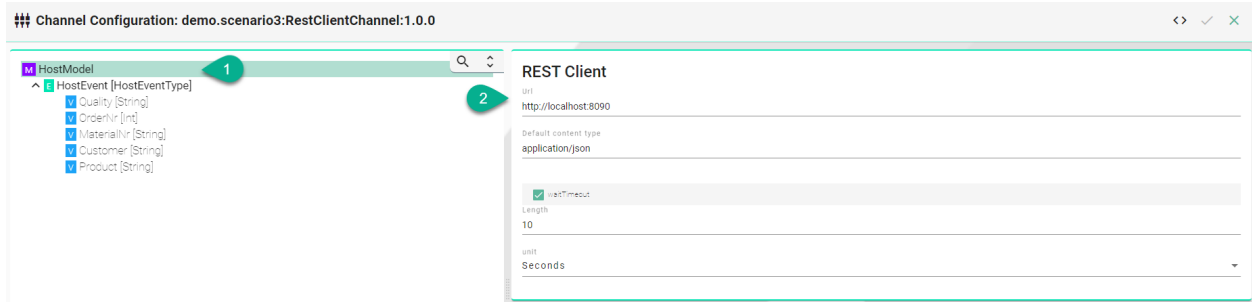
### Description of configuration properties:

Property	Description	Example
pathPrefix	Prefix for the URL	e.g., demo
Port	Port of the REST server	e.g., 9002, 9000, ...
IP	IP address of the REST server	http://localhost
DefaultContentType	Used to indicate the media type of the resource	application/json, application/xml, text/html, text/csv
webapp	Possibility to host an application	true, false

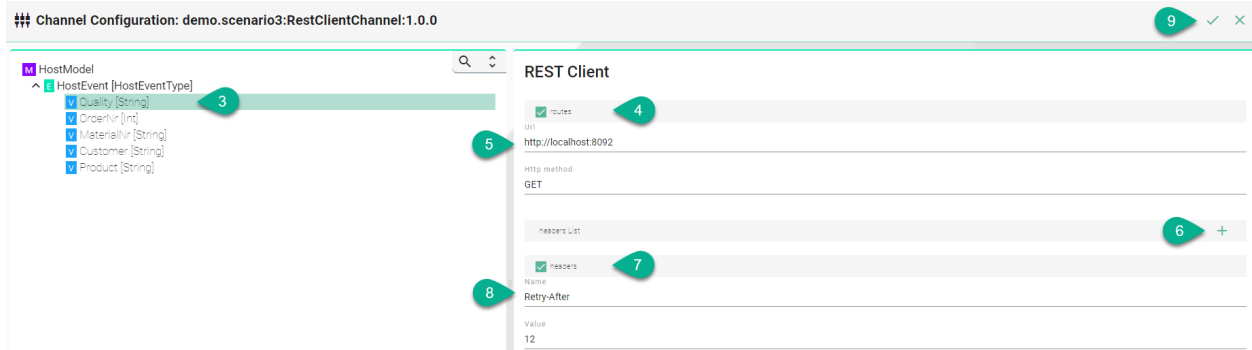
### REST Client

1. Select the **root model node** in the tree on the left.
2. Configuration of the REST Client
  - Enter the **IP**.
  - Enter the **Default-Content type**.
  - Enter a **timeout**.





3. Click on the **variable** in the tree which needs a configuration.
4. Check the **checkbox**.
5. Variable configuration.
  - Enter the **URL**.
  - Select the **HTTP method**.
6. Click on the **Add Header** button to add headers.
7. Check the **headers checkbox**.
8. Configure Key-Value pair of the header.
  - Enter a **name**.
  - Enter a **value**.
9. Click the **Apply** button and save the Channel by clicking the **Save** button on the upper right corner.



**Description of configuration properties:**

Property	Description	Example
URL	URL of the REST Server.	localhost:8090
RouteModelPath	Path of a node in the Information Model. A node can be a command, an event or a variable	/Model/ RestMetrics/Event/ partQualityEvent
RouteUrl	URL of the exposed node .	localhost:8090
HttpMethod	HTTP method for the action performed by the Client.	GET, POST, PUT
HeaderName and Header Value	To provide server and client with additional information	Retry-After: 12
ContentType	Is used to indicate the media type of the resource.	application/json
WaitTimeoutDuration	Timeout in seconds until request is failing	10

## 2.3 Mappings

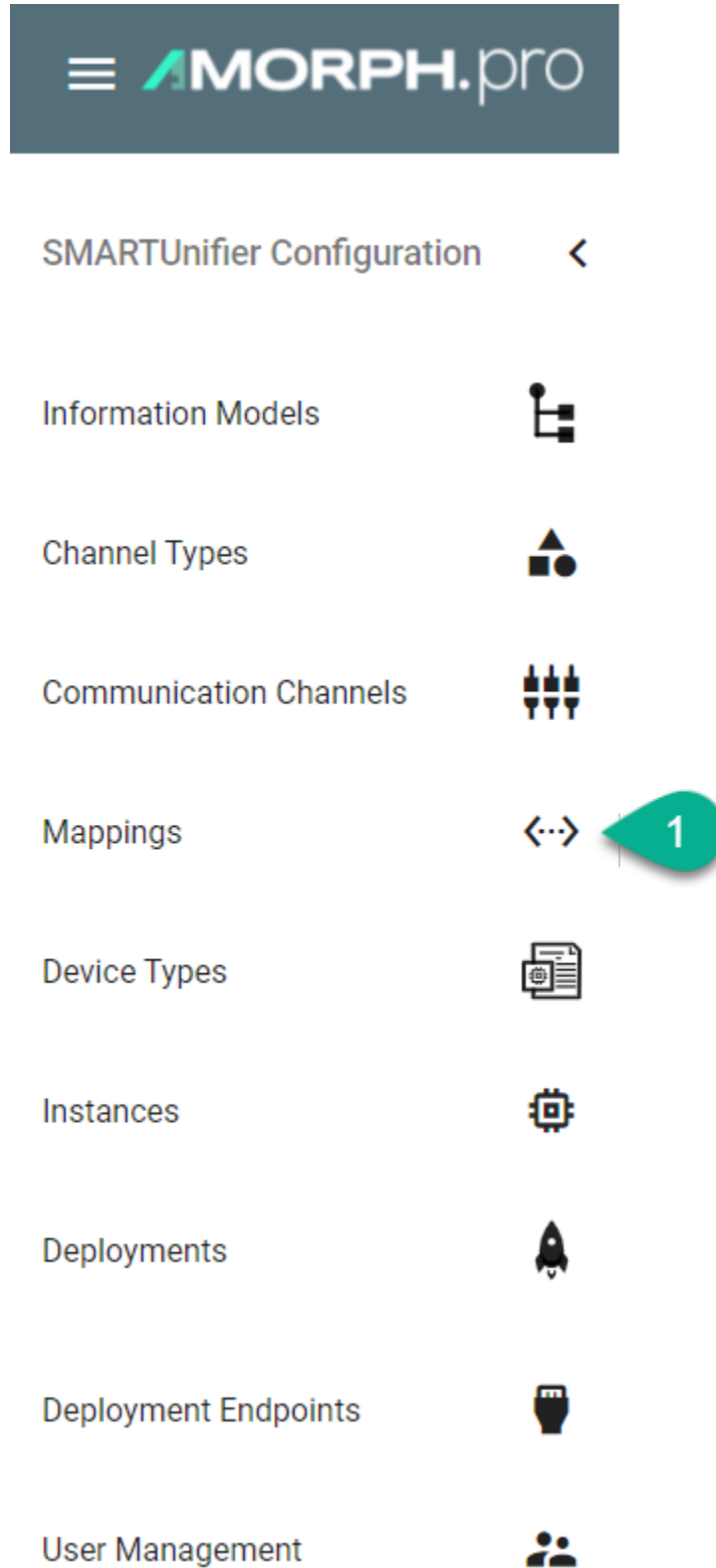
### 2.3.1 What are Mappings

Mappings represent the SMARTUNIFIER component that define when and how to exchange/transform data between two or multiple Information Models. In other words, it is acting as a translator between the different Information Models. One Mapping consists of one or multiple Rules. A Rule contains a Trigger, which defines when the exchange/transformation takes place, and a list of actions that are defining how the exchange/transformation is done.

### 2.3.2 How to create a new Mapping

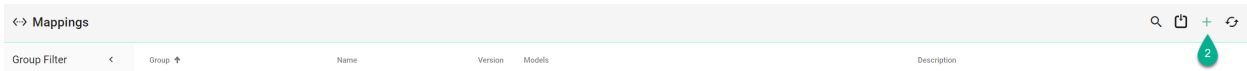
Follow the steps below to create a new Mapping definition:

- Go the Mappings perspective by clicking the “Mappings” button (1)

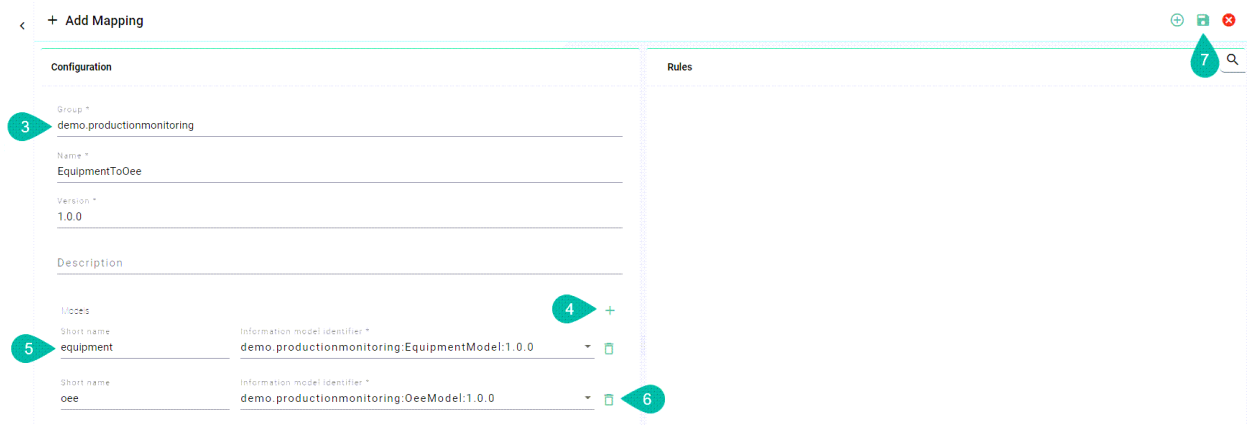


- Following screen containing a list view of existing Mappings is displayed

- In order to add a new Mapping, select the “Add Mapping” button at the top right corner (2)



- On the following screen provide the following mandatory information: Group, Name, Version and a Description which is optional (3)
- Click the “Add Model” button (4)
- Select the Information Model for this Mapping and enter a name for it (5)
- “Remove Model” button (6) removes the Model
- After all mandatory fields are filled in, the “Save” button at the top right corner is enabled. Click the button to submit the new Channel (7)
- The newly created Mapping is now visible in the list view

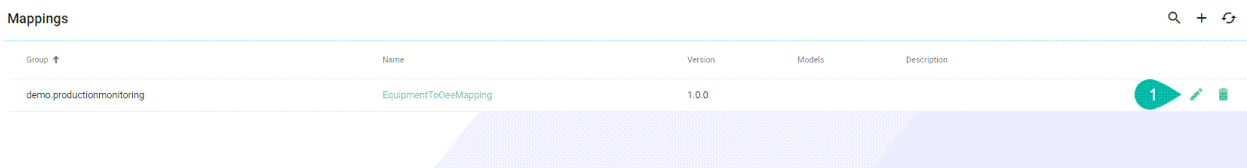


### 2.3.3 How to create Rules

#### Graphical

Follow the steps described below to create Rules:

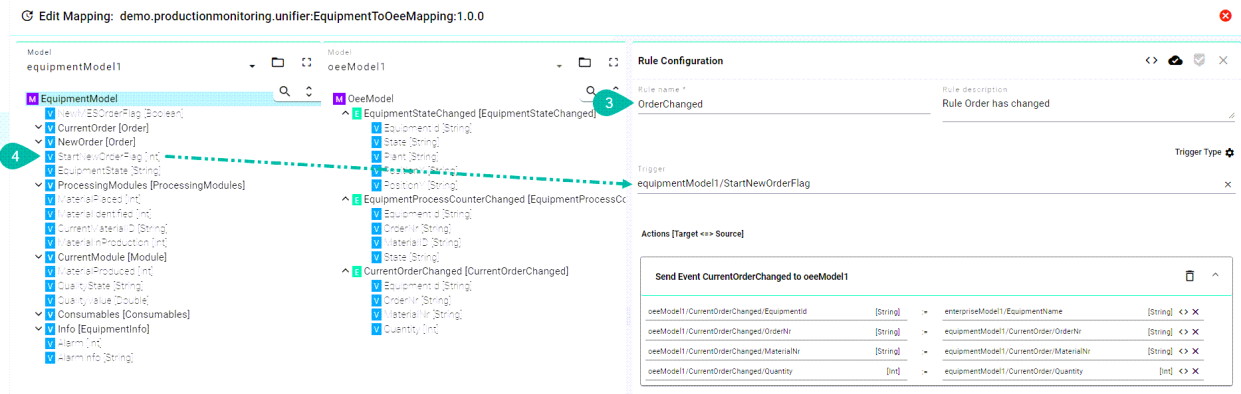
- Select the “Edit” button (1) placed in line with the mapping entry for which the mapping rule is to be created.



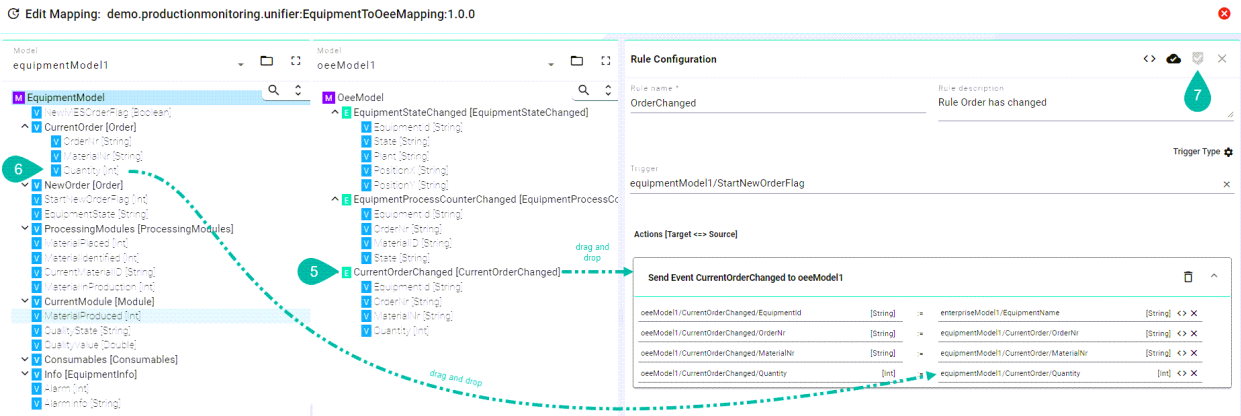
- The edit mapping view is displayed (see figure below):
- Select the “Add Rule” button at the top right corner (2).



- The following screenshot shows the Rule Editor.
- Enter Rule name (3).
- Drag and drop the Trigger from the model panes into the trigger field (4).



- Drag and drop the Source information into the Source field (5).
- Drag and drop the Target information from the model panes into the target field (6). The source field is enabled. The Source and the Target information types must be matched one on one (e.q., String to String)
- After all mandatory fields have been filled out, select the “Apply” button (7) in order to save the newly created Rule.
- The Rule Editor is closed and the newly created Rule is displayed in the Rules List.
- Select the “Save” button placed in the upper right corner to save the Mapping.



Rules Scenarios

A Rule is defined by its elements: Trigger, Target and Source. Each element is a note assigned from an Information Model.

Based on the combinations of a Rule elements, all the scenarios are listed in the table below.

Trigger	Target	Source	
Variable of a custom type	Variable	Variable	
		Variable of a custom type	
	Custom type Variable	Variable	
		Variable of a custom type	
	Variable of a custom type	Variable	
		Variable of a custom type	
	Event	Variable	
		Variable of a custom type	
	Command	Variable	
		Variable of a custom type	
	Variable	Variable	Variable
			Variable of a custom type
Custom type Variable		Variable	
		Variable of a custom type	
Variable of a custom type		Variable	
		Variable of a custom type	
Event		Variable	
		Variable of a custom type	
Command		Variable	
		Variable of a custom type	
Array of a custom type		Variable	Variable
			Variable of a custom type
	Custom type Variable	Variable	
		Variable of a custom type	
	Variable of a custom type	Variable	
		Variable of a custom type	
	Event	Variable	
		Variable of a custom type	
	Command	Variable	
		Variable of a custom type	
	Array	Variable	Variable
			Variable of a custom type
Custom type Variable		Variable	
		Variable of a custom type	
Variable of a custom type		Variable	
		Variable of a custom type	
Event		Variable	
		Variable of a custom type	
Command		Variable	
		Variable of a custom type	

Continued on next page

Table 3 – continued from previous page

Trigger	Target	Source
		Variable of a custom type
Property of a custom type	Variable	Variable
		Variable of a custom type
	Custom type Variable	Variable
		Variable of a custom type
	Variable of a custom type	Variable
		Variable of a custom type
	Event	Variable
		Variable of a custom type
	Command	Variable
		Variable of a custom type
Property	Variable	Variable
		Variable of a custom type
	Custom type Variable	Variable
		Variable of a custom type
	Variable of a custom type	Variable
		Variable of a custom type
	Event	Variable
		Variable of a custom type
	Command	Variable
		Variable of a custom type
Command	Variable	Variable
		Variable of a custom type
		Variable of a Command
	Custom type Variable	Variable
		Variable of a custom type
		Variable of a Command
	Variable of a custom type	Variable
		Variable of a custom type
		Variable of a Command
	Event	Variable
		Variable of a custom type
		Variable of a Command
Command	Variable	
	Variable of a custom type	
	Variable of a Command	
Event	Variable	Variable of a custom type
		Variable of an Event
		Variable
	Custom type Variable	Variable of a custom type
		Variable of an Event
		Variable
	Variable of a custom type	Variable of a custom type
Variable of an Event		
Variable		

Continued on next page

Table 3 – continued from previous page

Trigger	Target	Source
	Event	Variable of a custom type
		Variable of an Event
		Variable
	Command	Variable of a custom type
		Variable of an Event
		Variable

## Code

More complex scenarios, which are currently not supported by the graphical view can be implemented via the code editor. Rules are based on Scala.

### Basics - Rule construct

A Rule is always starting with a Trigger (1). The Trigger can represent a Variable, an Event or a Command; within one of the selected Information Models. After the trigger call `mapTo` (2) and define the function body by adding curly braces (3). Depending on the Trigger declare the `TriggerInstance` (4). Depending on the type of the Trigger use the naming accordingly:

```

1 Trigger mapTo { TriggerInstance =>
2
3
4
5
}
```

The Source (5) is the content of the `TriggerInstance` (e.g., In case the Trigger is a Variable, then is the Source an Instance of that Variable) In order to assign the Source to the Target, add the `:=` operator (6). The Target can be any variable you want to map to (7).

```

Trigger mapTo { TriggerInstance =>
7 Target := Source
6
}
```



## Variable to Event Mapping

In this case the mapping of the Complex Variable *CurrentOrder* in the *EquipmentModel* and of a Simple Variable in the *EnterpriseModel* to the *EquipmentNewOrderStart* Event in the *MesModel* is described.

- Trigger: **EquipmentModel.StartNewOrderFlag** (line 1)
- TriggerInstance of *EquipmentModel.Alarm*: **variable** (line 1)
- Since values are assigned to an Event, call the function - *send*, on the *EquipmentNewOrderStartEvent* (line 2) and define the TriggerInstance - **event** (line 2).
- The Targets are defined by entering the path of the variables in the event - **event.EquipmentId** (line 3).

Listing 1: Rule - StartOrder - Variable/Event

```

1 EquipmentModel.Alarm mapTo {variable =>
2   MesModel.EquipmentAlarm.send(event => {
3     event.EquipmentId := EnterpriseModel.EquipmentName
4     event.OrderNr := EquipmentModel.CurrentOrder.OrderNr
5     event.MaterialID := EquipmentModel.CurrentMaterialID
6     event.AlarmInfo := EquipmentModel.AlarmInfo
7     CommunicationLogger.log(variable, event)
8   })
9 }
```

## Event to Variable Mapping

In this case the mapping of values inside the *TransferNewOrder* Event from the *MesModel* into variables from the *EquipmentModel* is described.

- The Trigger is defined by entering the path of the Event - **MesModel.TransferNewOrder** (line 1). Since an Event is used as Trigger, the TriggerInstance is named accordingly - **event** (line 1).
- In the function body provide the Complex Variable *NewOrder* and the Simple Variable *NewMESOrderFlag* with data from the *MesModels TransferNewOrder* Event.
- Targets are defined by entering the path of the variables like - **EquipmentModel.NewOrder.OrderNr** (line 2).
- In order to assign values to *OrderNr*, *MaterialNr* and *Quantity* of the Complex Variable *NewOrder*, enter the TriggerInstance event followed by the variable name of the *TransferNewOrder* Event - **event.OrderNr** (line 2).
- In this case it is also possible to provide the variable *NewMesOrderFlag* with a Boolean like - **true** (line 5).

Listing 2: Rule - TransferNewOrder - Event/Variable

```

1 MesModel.TransferNewOrder mapTo {event =>
2   EquipmentModel.NewOrder.OrderNr := event.OrderNr
3   EquipmentModel.NewOrder.MaterialNr := event.MaterialNr
4   EquipmentModel.NewOrder.Quantity := event.Quantity
5   EquipmentModel.NewMESOrderFlag := true
6 }

```

## Commands Mapping

The following scenario describes a Rule mapping incoming data from a file to MQTT. When the *FileEvent* is triggered - the rule executes first the *DatabaseCommand* in order to retrieve data from a database.

- Trigger is defined by entering the path of the Event - **file.FileEvent** (line 1). Since an Event is used as Trigger, the TriggerInstance should be named accordingly - **event** (line 1).
- Inside the function body execute a Command. The execution of a Command is defined by entering the path of the Command. At the end of the path, call the **execute** function (line 2). The TriggerInstance is named accordingly - **command** (line 3).
- The lines 3-5 show the first part of the Command. Here assign values from the source model to the Command Parameters.
- Since every Command has a Reply, we need to define the reply section - (line 6).
- In this case send out the data over MQTT after the data is retrieved from the database. In the reply function body, enter the path of the *MqttEvent*. Since this is the 2nd Event, the TriggerInstance can be named - **event1** (line 1).
- Inside the function body assign values from the *FileEvent* (line 8-10) as well as from the Reply (line 11-12) to the *MqttEvent*.

Listing 3: Rule - File2MqttWithDB - Event/Commands

```

1 file.FileEvent mapTo {event =>
2   database.DatabaseCommand.execute(command => {
3     command.orderNr := event.orderNr
4     command.materialNr := event.materialNr
5     CommunicationLogger.log(event, command)
6   }, reply => {
7     mqtt.MqttEvent.send(event1 => {
8       event1.Quality := event.quality
9       event1.OrderNr := event.orderNr
10      event1.MaterialNr := event.materialNr
11      event1.Customer := reply.customer
12      event1.Product := reply.product
13      CommunicationLogger.log(reply, event1)
14    })

```

(continues on next page)

(continued from previous page)

```

15   })
16   }

```

## Mapping with Lists

The following scenario describes a Rule that is mapping incoming data from a file to MQTT. The MQTT Model contains a List called *DataList*. **Note** that lists can only be mapped in the code view.

- Create a variable *listItem* that holds a reference of a *newItem* in the *DataList* (line 6)
- Call the variable from the *listItem* and assign the value from the file event (line 8)

Listing 4: Rule - FileToMQTT - Lists

```

1  csv.FileEvent mapTo { event =>
2
3      event.items.foreach { item =>
4          mqtt.MqttEvent.send(event1 => {
5
6              val listItem = event1.DataList.newItem
7
8              listItem.Timestamp := item.Timestamp
9              listItem.Pressure := item.Alarmlevel
10
11             CommunicationLogger.log(event, event1)
12         })
13     }
14 }

```

## Logging

Logging can be added in the Rule implementation by calling - **CommunicationLogger.log** (line 4)

Listing 5: Rule with Logging

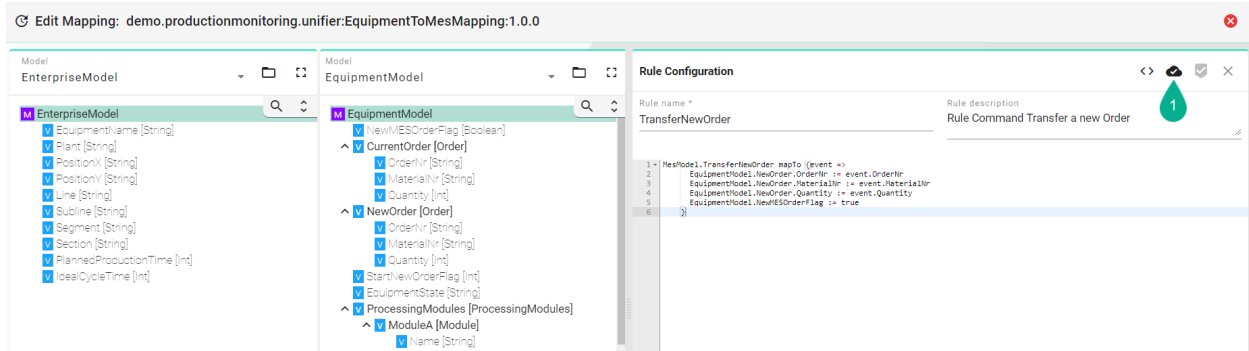
```

1  EquipmentModel.Alarm mapTo {variable =>
2      MesModel.EquipmentAlarm.send(event => {
3          event.EquipmentId := EnterpriseModel.EquipmentName
4          CommunicationLogger.log(variable, event)
5      })
6  }

```

## Compiling

You can compile the code for the selected Rule by clicking the “Compile” button (1) and check for compilation errors before saving the Rule.



## 2.4 Device Types

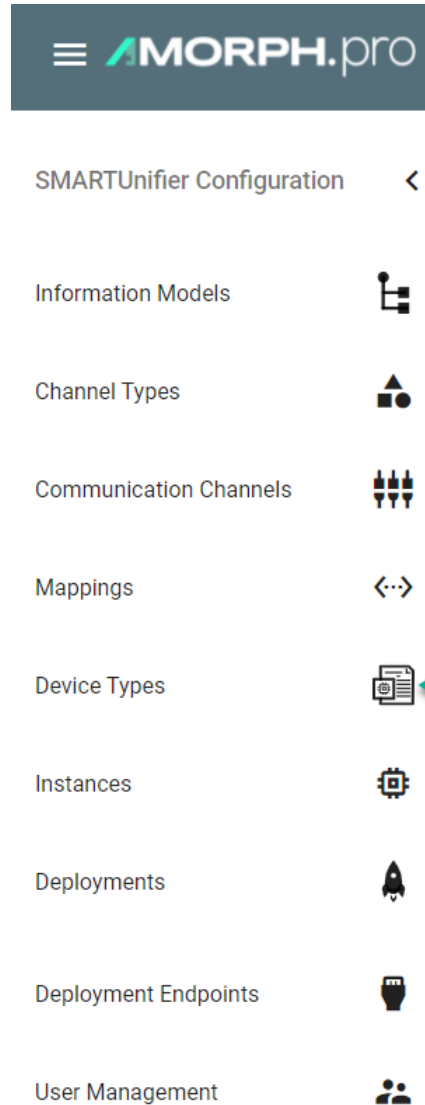
### 2.4.1 What are Device Types

With SMARTUNIFIER Device Types it is possible to have multiple Communication *Instances*, which share common configuration parameters. A Device Type contains one or multiple Mappings. Each Mapping contains one or multiple Information Models and its associated Communication Channel. Within a SMARTUNIFIER Device Type it is possible to over-write existing Communication Channel configurations. Device Types are especially important, when integrating several similar pieces of equipment or devices. In this case, the Device Type can be reused for all Instances (i.e., one instance represents one equipment).

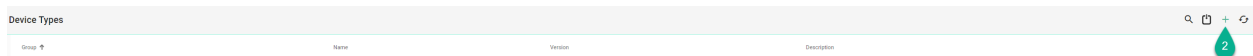
### 2.4.2 How to create a new Device Type

Follow the steps described below to create a SMARTUNIFIER Device Type.

- Select the SMARTUNIFIER Device Type Perspective (1).



- Click on the “Add Device Type” button from the upper right corner (2).



- The creation of a Device Type is split up into two parts. First provide the basic information about the Device Type like the Group, the Name, and the Version. Optionally, provide a short description (3).
- In the next step provide one or multiple *Mappings* previously created. To do so click the “Add Mapping” button (4). After selecting a Mapping (5) the associated Information Models show up. In case the wrong Mapping was selected click the “Delete Mapping” button to remove the Mapping from the Device Type (6). Now select a *Communication Channel* for each *Information Model* from the Drop-Down (7).
- Similar to the Communication Channel view it is possible to change the configuration of the Channel within the Device Type view. In case of changes in the configuration click the “Configure” button (8). This action over-writes previous configurations.

- The new Device Type can be saved by clicking the “Save” button at the top right corner (9).

⊕ Add Device Type 9

---

3 Group \*  
demo.productionmonitoring

Name \*  
SUDeviceType

Version \*  
1.0.0

Description

Mappings 4 +

5 Mapping  
demo.productionmonitoring.unifier:EquipmentToOeeMapping:1.0.0 6

Models	Channels
enterpriseModel1 <span style="float: right;">7</span>	demo.productionmonitoring.unifier:EnterpriseChannel:1.0.0 <span style="float: right;">8</span>
equipmentModel1	demo.productionmonitoring.unifier:EquipmentOpcUaClientChannel:1.0.0
oeeModel1	demo.productionmonitoring.unifier:OEEQtChannel:1.0.0

## 2.5 Instances

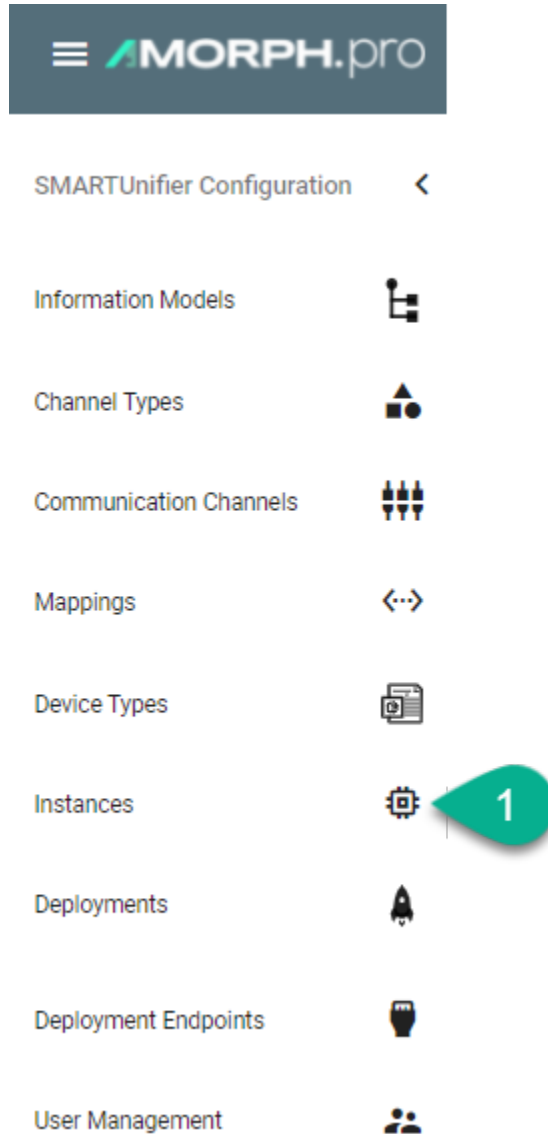
### 2.5.1 What are Instances

A SMARTUNIFIER Instance is a dynamically created application that can be deployed to any suitable IT resource (e.g., Equipment PC, Server, SMARTUNIFIERBox, Cloud), and which provides the connectivity functionality configured. Therefore, a SMARTUNIFIER Instance uses one or multiple Mappings and selected Communication Channels from a previously defined *Device Type*.

### 2.5.2 How to create a new Instance

Follow the steps described below to create a SMARTUNIFIER Instance.

- Select the SMARTUNIFIER Instances Perspective (1).



- Click on the “Add Instance” button from the upper right corner (2).



- Select a Device Type from the Drop-Down (3)
- The details for the Instance are automatically taken from the Device Type (4). However, Group, Name, Version and the Description can still be changed.
- The Mapping defined in the Device Type show up in the Mapping area (5).
- To change the existing configuration or if no configuration has been made yet, click the “Configure” button (6)
- Save the SMARTUNIFIER Instance by clicking the “Save”(7)

⊖ Add Instance

- In order to deploy, run and stop the Instance navigate to the *Deployment* perspective.



## DEPLOYMENT

SMARTUNIFIER supports the *deployment* of Instances on several computing environments:

- *Local* - on the same environment the SMARTUNIFIER Manager is running on.
- *Docker* - on containerized environments.
- *Fargate* - on the AWS Cloud using fully managed service AWS Fargate.

Learn how to *operate* and *monitor* your SMARTUNIFIER Instances.

### 3.1 What is a Deployment

With the SMARTUNIFIER Deployment capability you can deploy your SMARTUNIFIER *Instances* to any IT resource (e.g., Equipment PC, Server, SMARTUNIFIERBox, Cloud) suitable to execute SMARTUNIFIER Instances.

Depending on the Deployment Type a Deployment Endpoint has to be initially created. For deployments on a local computer, no Deployment Endpoint needs to be set.

Currently, the following Deployment Endpoints are supported:

- *Local*: Deployment of a SMARTUNIFIER Instance to your local Computer where the SMARTUNIFIER Manager is running on.
- *Docker*: Deployment of a SMARTUNIFIER Instance using Docker Container
- *AWS*: Deployment of a SMARTUNIFIER Instance on the AWS Cloud using AWS Fargate.

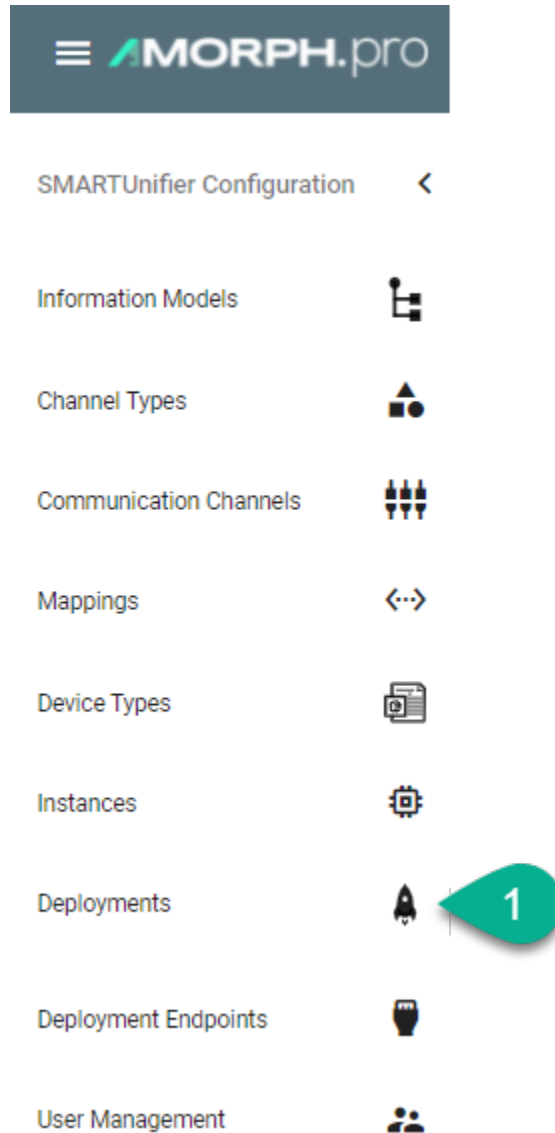
#### Getting started:

- Select your environment and create the Deployment:
  - *Local*
  - *Docker*
  - *Fargate*
- Learn how to *operate* an Deployment.
- Learn how to *monitor* an Deployment.

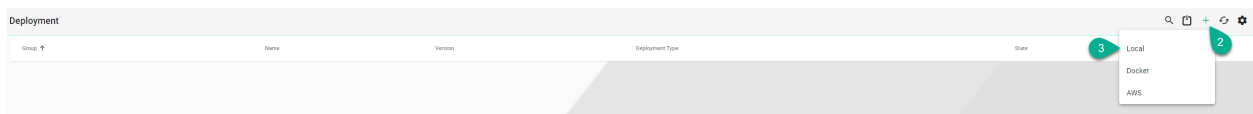
## 3.2 Deploy Locally

Follow the steps described below to order deploy an Instance locally:

- Select the SMARTUNIFIER Deployment perspective (1).



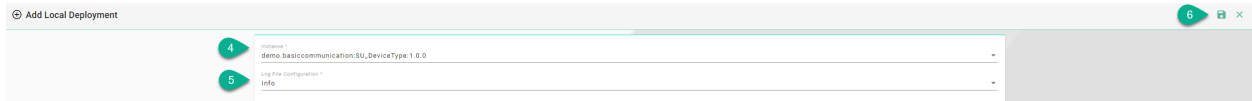
- Click on the “Add Deployment” button (2).
- Select the Deployment Type **Local** from the pop-up (3).



- Select the SMARTUNIFIER Instance to be used in the Deployment (4).
- Select the level for the log file configuration (5). We recommend the log level of type *Info* in

case of a normal deployment scenario.

- When all mandatory fields are filled click the “Save” button (6).



### 3.3 Deploy with Docker

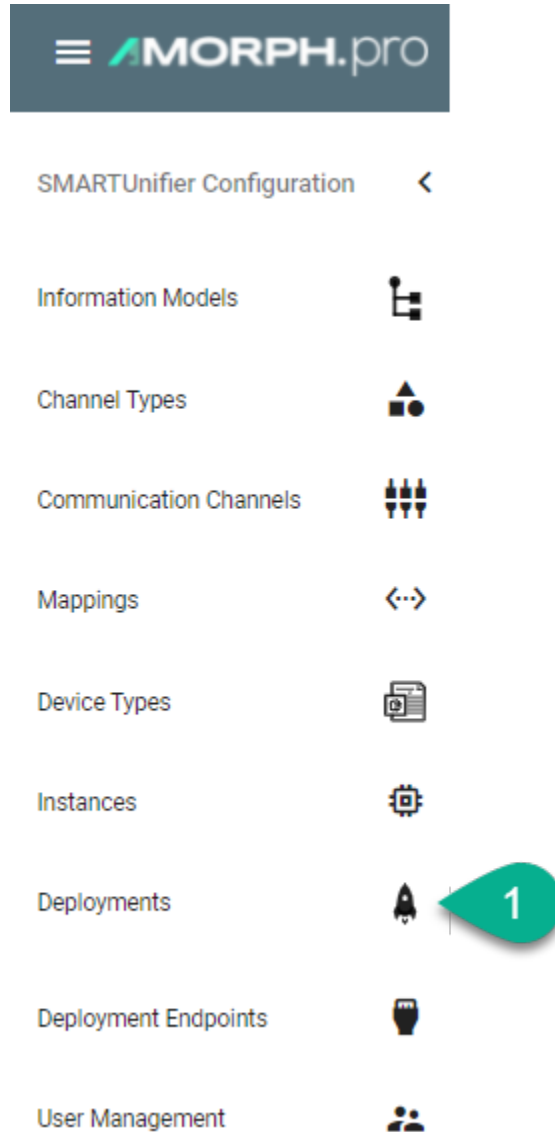
---

**Note:** Before creating the deployment of an Instance make sure you created a *Docker Deployment Endpoint*. The Deployment Endpoint specifies the location where you want the container to run.

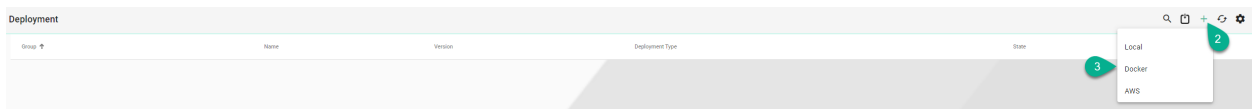
---

Follow the steps described below to order deploy an Instance using a Docker Container:

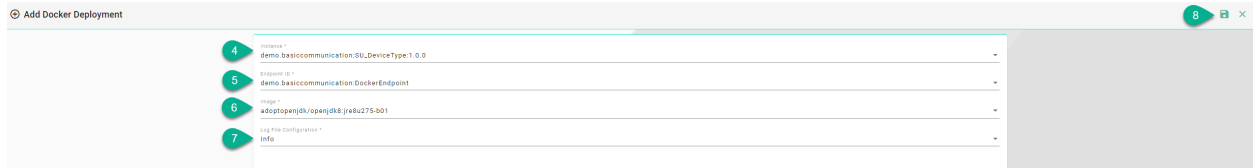
- Select the SMARTUNIFIER Deployment perspective (1).



- Click on the “Add Deployment” button (2).
- Select the Deployment Type **Docker** from the pop-up (3).



- Select the SMARTUNIFIER Instance to be used in the Deployment (4).
- Select the Docker Endpoint ID created in the [Docker section](#) from the Drop-Down menu (5).
- Select the Image from the Drop-Down menu (6).
- Select the level for the log file configuration (7). We recommend the log level of type *Info* in case of a normal deployment scenario.
- When all mandatory fields are filled click the “Save” button (8).



## 3.4 Deploy with AWS Fargate

SMARTUNIFIER supports the deployment of Instances on Amazon Web Services (AWS) using AWS Fargate.

In order to deploy your SMARTUNIFIER Instances using AWS Fargate an AWS Account is required.

Before deploying a SMARTUNIFIER-Instance using AWS Fargate please refer to the [Prerequisites](#) section and make sure all requirements your Account needs to fulfill are met.

### 3.4.1 Prerequisites

#### Specialized Knowledge

Before deploying and operating SMARTUNIFIER Instances using AWS Fargate, it is recommended that you become familiar with the following AWS services. (If you are new to AWS, see [Getting Started with AWS](#))

- [Amazon Elastic Container Service \(ECS\)](#)
- [Amazon Virtual Private Cloud \(VPC\)](#)
- [Amazon CloudWatch](#)

You should also be familiar with the used Communication Channel and its capabilities of the deployed SMARTUNIFIER Instance.

#### AWS Resources

For the deployment of SMARTUNIFIER Instances on AWS Fargate the following resources are required:

#### Amazon S3 - Bucket

SMARTUNIFIER is using an Amazon S3 Bucket to upload Instances in an archive file format. We recommend to [create a private Bucket](#) dedicated for the SMARTUNIFIER.

#### AWS VPC and Subnets

In order for SMARTUNIFIER to deploy Instances your AWS account a [VPC and Subnets](#) are needed. Please note that the Default VPC should not be used.

### Amazon ECS - Cluster

SMARTUNIFIER is using AWS Fargate for the deployment of Instances on the AWS Cloud. Therefore an ECS Cluster is required. We recommend to [create one Cluster](#) dedicated for SMARTUNIFIER deployed Instances.

### AWS ECR - Repository

SMARTUNIFIER is using an AWS ECR repository in order to push Docker Images, which is created by an AWS CodeBuild project. We recommend to [create one repository](#) dedicated for SMARTUNIFIER Instance images.

### IAM - User

SMARTUNIFIER complies with the [security best practices in IAM](#) and does not need root privileges. We recommend to [create one user](#) dedicated for SMARTUNIFIER. The IAM user follows the general rule of least privileges and allows only policies needed for the deployment of SMARTUNIFIER Instances.

Create the IAM user by following the steps described in the AWS IAM documentation [the IAM dashboard](#). The IAM user for SMARTUNIFIER must use the AWS access type **programmatically access**.

Attach the following permission:

Policy ARN	Description
<i>arn:aws:iam::aws:policy/AmazonS3FullAccess</i>	Provides full access to all buckets via the AWS Management Console.
<i>arn:aws:iam::aws:policy/AWSCodeBuildAdminAccess</i>	Provides full access to AWS CodeBuild via the AWS Management Console. Also attach AmazonS3ReadOnlyAccess to provide access to download build artifacts, and attach IAMFullAccess to create and manage the service role for CodeBuild.
<i>arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryFullAccess</i>	Provides administrative access to Amazon ECR resources.
<i>arn:aws:iam::aws:policy/AmazonECS_FullAccess</i>	Provides administrative access to Amazon ECS resources and enables ECS features through access to other AWS service resources, including VPCs, Auto Scaling groups, and CloudFormation stacks.
<i>arn:aws:iam::aws:policy/CloudWatchFullAccess</i>	Provides full access to CloudWatch.

### Programmatic system credentials

SMARTUNIFIER needs the set up of a [credential profile](#) in order to deploy Instances on AWS Fargate. We recommend to [create a new access key](#) after 90 days.

Listing 1: Credentials Profile

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

### IAM Role - AWS CodeBuild Service Role

CodeBuild requires a service in order to interact with dependent AWS services:

- Access to Amazon S3 in order to retrieve SMARTUNIFIER Instance artifacts - such as libraries and configuration files.
- Access to AWS ECR in order to push the container image in the specified repository

Create the following [IAM Role](#) via the AWS console.

Listing 2: AWS CodeBuild Service Role

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CodeCommitPolicy",
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3GetObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3PutObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECRPullPolicy",
      "Effect": "Allow",
      "Action": [

```

(continues on next page)



(continued from previous page)

```

        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "ECRAuthPolicy",
    "Effect": "Allow",
    "Action": [
        "ecr:GetAuthorizationToken"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "S3BucketIdentity",
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
    ],
    "Resource": "*"
}
]
}

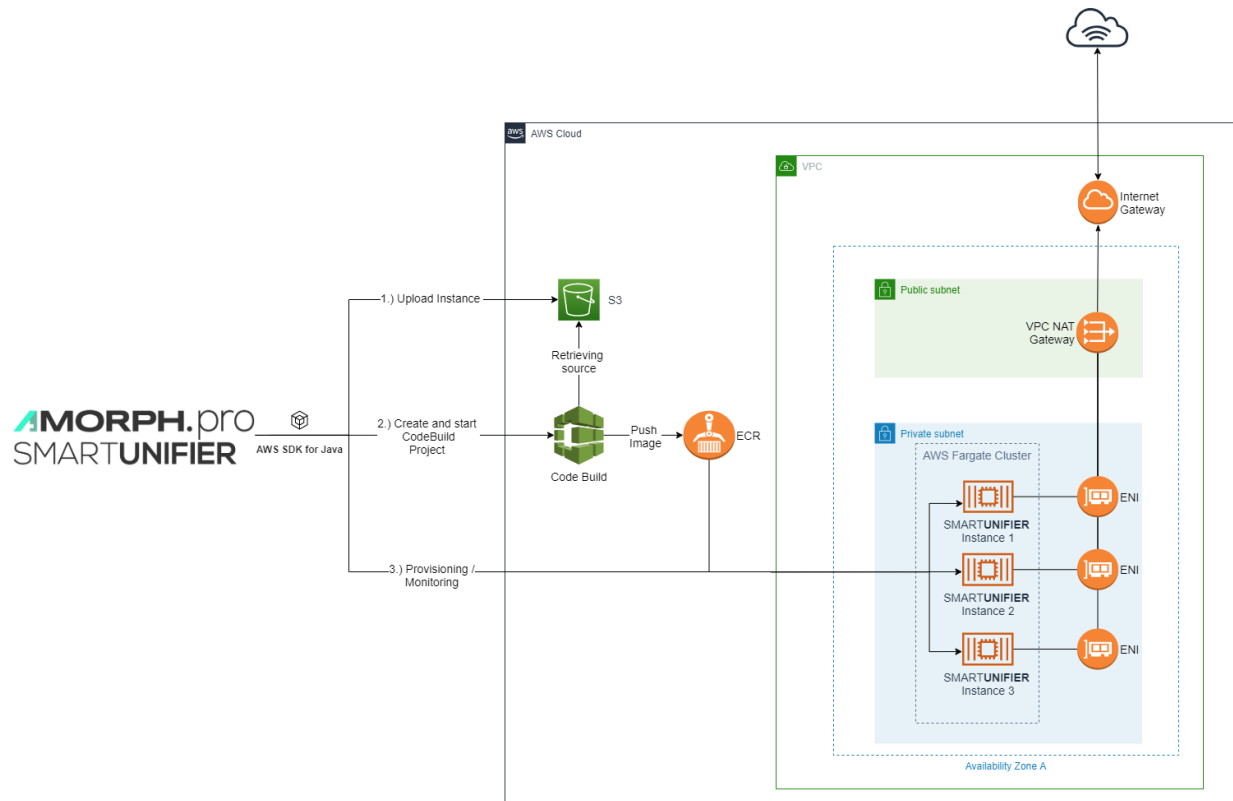
```

### 3.4.2 Architecture

The deployment of SMARTUNIFIER-Instances on AWS Cloud is handled by the SMARTUNIFIER Manager. The Manager can run on any On-Premise location such as, server environments and Industrial PCs; however, in order to deploy Instances on AWS an internet connection is required. In order to run SMARTUNIFIER Manager on AWS Cloud please see the SMARTUNIFIER Installation Manual.

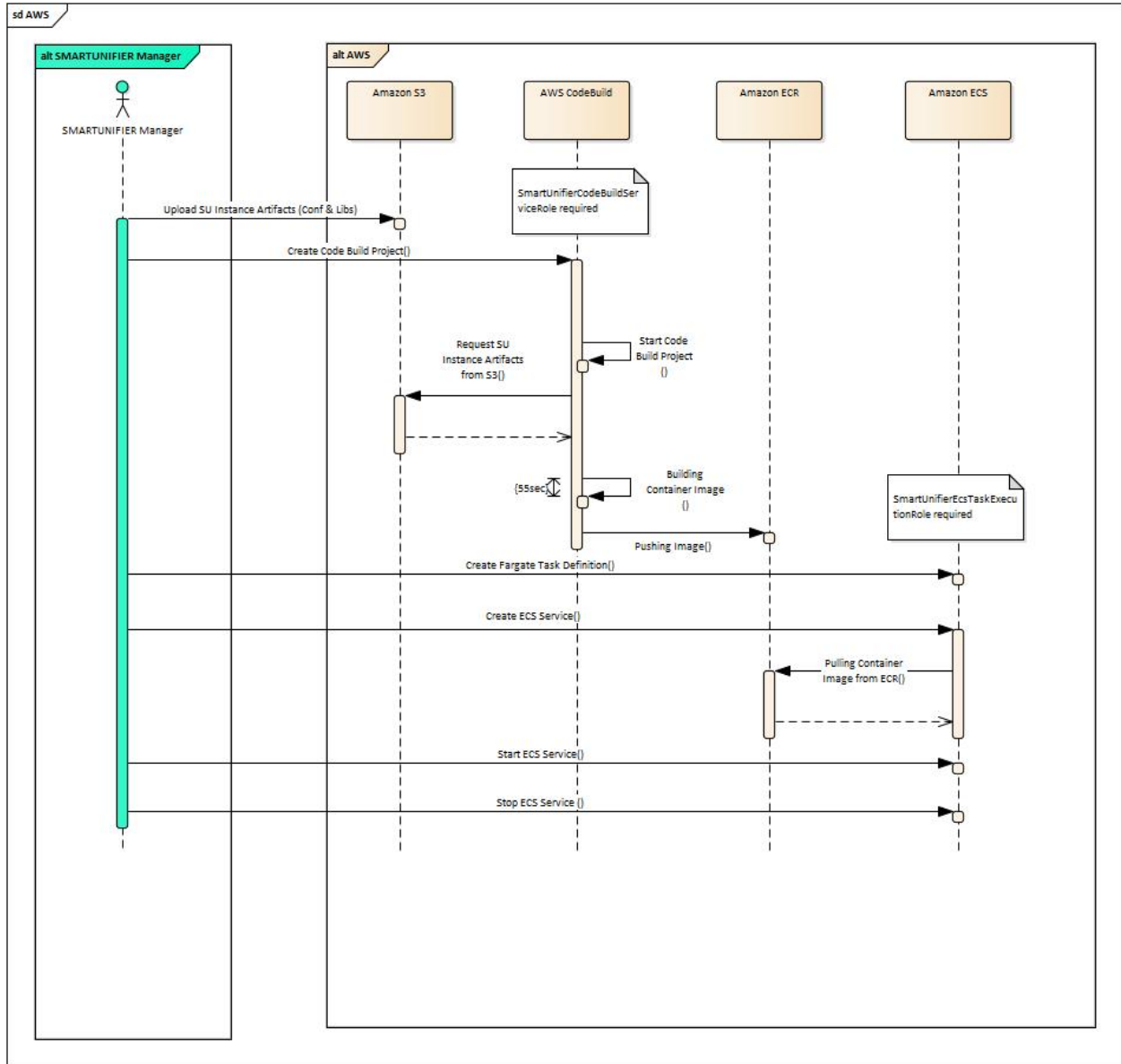
SMARTUNIFIER is using the [AWS SDK for Java](#) to make deployments of Instances to [AWS Fargate](#). Following AWS Services are used during the deployment process:

- [AWS Simple Storage Service \(Amazon S3\)](#) (Mandatory).
- [AWS CodeBuild](#) (Mandatory).
- [AWS Elastic Container Registry](#) (Mandatory).
- [AWS Elastic Container Service](#) (Mandatory).
- [AWS Fargate](#) (Mandatory).
- [Amazon CloudWatch](#) (Optional).



## Sequence of events

1. Upload of the SMARTUNIFIER Instance as an archive file format to Amazon S3.
2. Creation and automatic triggering of an AWS CodeBuild project.
3. The AWS CodeBuild project uses the archive file from the specified Amazon S3 Bucket in order to build a Docker Image for the particular SMARTUNIFIER Instance.
4. When finished, AWS CodeBuild pushes the Image to a specified ECR Repository.
5. Is the Image available on the ECR Repository a Fargate Task Definition is created as well as an ECS Service which is using the Task Definition.
6. By default, the Task is not started directly. Starting and Stopping of tasks can be done via the SMARTUNIFIER Manager or the AWS Console.



### 3.4.3 Planning the Deployment

#### Task Sizing

Each SMARTUNIFIER Instance runs as java byte code, thus having a low footprint. We recommend using the following guideline for Task Sizing.

**Note:** Please note that AWS Fargate is pricing based on the vCPU and memory resources, which are specified during the set up.

CPU	Memory Values	Instance (Number of Mappings)	Workload of Map-pings)
0.25 vCPU	0.5GB, 1GB, and 2GB	<= 5	
0.5 vCPU	Min. 1GB and Max. 4GB, in 1GB increments	> 6	

### 3.4.4 Deployment Steps

#### Expected Time

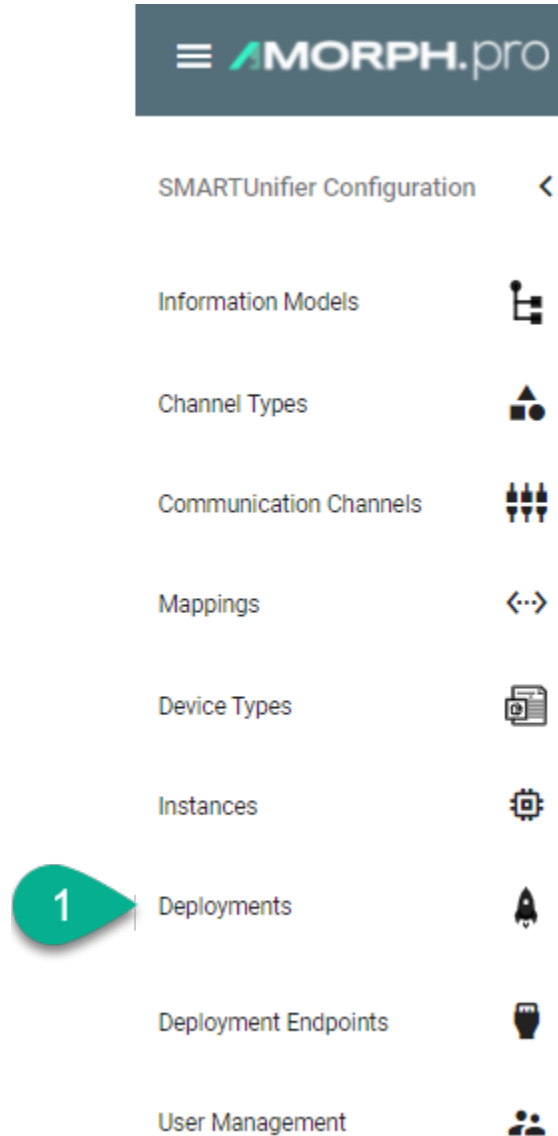
- Deployment of an SMARTUNIFIER Instance on AWS Fargate (Existing AWS Resources) - expected deployment time: **5 min**
- Deployment of an SMARTUNIFIER Instance on AWS Fargate (Creation of needed AWS Resources required) - expected deployment time: **30-40 min**

#### Deployment of the SMARTUNIFIER Instance

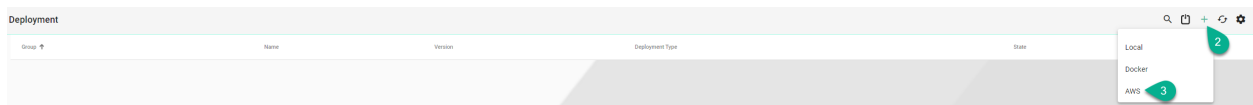
If you have not already set up an AWS Deployment Endpoint please refer to chapter: [AWS Endpoint](#).

Follow the steps described below to deploy a SMARTUNIFIER Instance on AWS Fargate:

- Select the SMARTUNIFIER Deployment perspective (1).

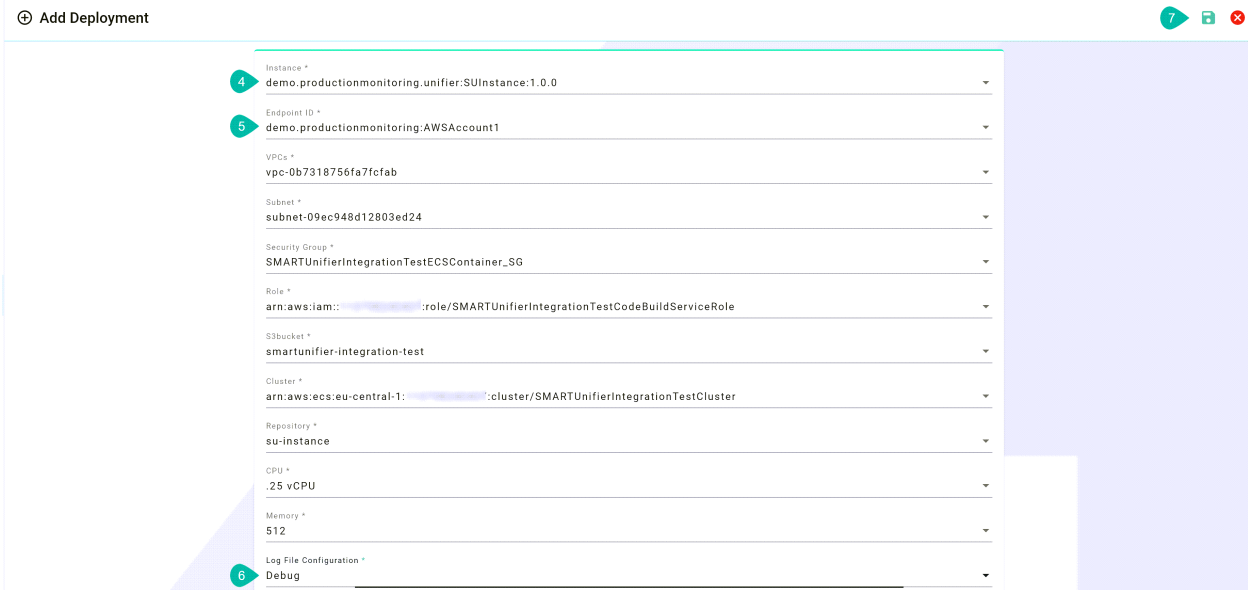


- Click the “Add” button (2).
- Select AWS (3).

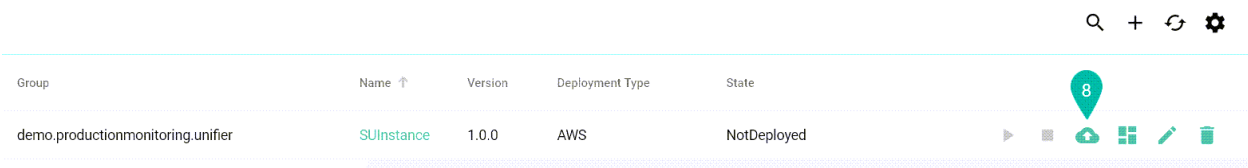


- Select the SMARTUNIFIER Instance you want to deploy (4):
- Select your AWS account in form of a Deployment Endpoint created *previously* (5) and configure the following parameters:
  - Select the **VPC** in which you want to deploy the SMARTUNIFIER Instance.
  - Select a **Subnet** within the VPC.
  - Select a **Security Group**.

- Select a **IAM Role** for AWS CodeBuild.
  - \* AWS CodeBuild needs a service role so that it can interact with dependent AWS services on behalf of SMARTUNIFIER.
- Select a **S3 Bucket**.
- Select a **ECS Cluster** in which the Instance should be deployed.
- Select an **ECR Repository**.
  - \* The AWS CodeBuild project, which is created and triggered by SMARTUNIFIER, pushes an Image to the provided Amazon ECR Repository.
- Select the *Task's* - **CPU**.
- Select the *Task's* - **Memory**.
- Select the **Log File Configuration** (Determines the log level detail) (6).
- Save the Deployment by clicking the “Save” button (7).



- Go back to the list view by clicking the “Close” button and deploy your SMARTUNIFIER Instance by clicking the “Deploy” button (8).



- You can start and stop the Instance using SMARTUNIFIER by clicking the “Start”/”Stop” button or using the AWS Console.

## Monitoring

Once deployed and started, the SMARTUNIFIER Instance logs can be accessed via Amazon CloudWatch.

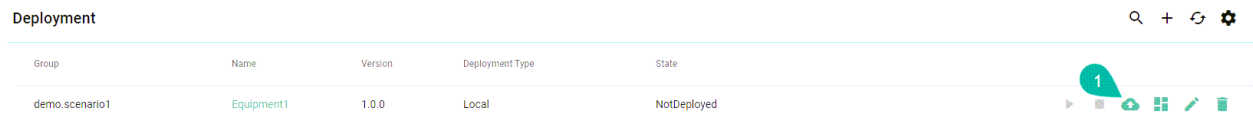
In order to access log files follow the steps below:

- Go to the Amazon CloudWatch Service via the Console.
- Select **Log groups** from the menu on the left.
- Select **awslogs-testinstance** and select a log Stream.

## 3.5 How to deploy, run and operate a deployed Instance

### 3.5.1 How to deploy an Instance


- In order to start the Instance, click first the “Deploy” button (1). A message is shown, that confirms the successful deployment of the Instance.



Group	Name	Version	Deployment Type	State
demo.scenario1	Equipment1	1.0.0	Local	NotDeployed

### 3.5.2 How to run an Instance


- After successfully deploying the Instance, the state changes from *NotDeployed* to *Stopped*. You can now click the enabled “Start” button (2). The Instance state will change to *Started*. A message is shown, that confirms the successful start of the Instance.



Group	Name	Version	Deployment Type	State
demo.scenario1	Equipment1	1.0.0	Local	Stopped

### 3.5.3 How to stop an Instance

- To stop the Instance, click the “Stop” button (3).



Group	Name	Version	Deployment Type	State
demo.scenario1	Equipment1	1.0.0	Local	Started

### 3.5.4 How to delete a Deployment of an Instance

- Click on the “Delete” button to delete the Deployment for a specific Instance (4). This is only possible if the Instance is in the state *Stopped*.
- Click on the “Edit” button to perform changes to the Deployment (5). It is only possible to edit a Deployment if the Instance is not yet deployed. In the case the Instance is already deployed, only the details of the Deployment can be viewed.

Deployment			+ ↻
Instance	Deployment Type	State	
demo.scenario1.Equipment1:1.0.0	Docker	Started	▶ ⏸ ⏹ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷

### 3.5.5 How to un-deploy an Instance

- In order to un-deploy an Instance, **make sure** that the Instance is not running. If necessary *stop the Instance*.
- Go to the edit Deployment view by clicking the “Edit” button.
- Click the “Remove Deployment” button in the upper right corner (6).
- The Instance state changes to *NotDeployed* and the Deployment can be edited. Please **note** that the Instance associated with the Deployment cannot be changed.

Deployment					🔍 + ↻ ⚙️
Group	Name	Version	Deployment Type	State	
demo.scenario1	Equipment1	1.0.0	Local	Stopped	▶ ⏸ ⏹ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷

## 3.6 How to monitor a deployed Instance

- In order to monitor an Instance, access the Dashboard view by clicking the “Dashboard” button (7).
- If the Instance is in the state *NotDeployed* the Dashboard cannot be accessed.

Deployment					🔍 + ↻ ⚙️
Group	Name	Version	Deployment Type	State	
demo.scenario1	Equipment1	1.0.0	Local	Started	▶ ⏸ ⏹ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷

- The Dashboard provides the following information:
  - *Channels* associated with the Instance
  - *Mappings* associated with the Instance
  - CPU Usage of the Instance
  - Memory Usage of the Instance



- Status of the Instance
- Start time of the Instance

**Status Dashboard: demo.scenario1:Equipment1:1.0.0**

Channels			
Info	Type	Status	Model
demo.basiccommunication.CSVChannel:1.0.0	layer-csvstring2model	Connected	demo.basiccommunication.CsvDataModel:1.0.0
demo.basiccommunication.RESTServerChannel:1.0.0	implementation-rest-server	Connected	demo.basiccommunication.RestDataModel:1.0.0

Mappings	
Info	Models
demo.basiccommunication:CSVToRESTMapping:1.0.0	CsvDataModel, RestDataModel

**Status** Started

**Time started** 2021-02-11 18:14:11

**Time Up** 0:01:12

**CPU Usage** (0%)

**Memory Usage** (1.23%)

## ADMINISTRATION

Learn how to create *Deployment Endpoints* and how to manage *User Accounts* within SMARTUNIFIER.

### 4.1 Deployment Endpoints

#### 4.1.1 What are Deployment Endpoints

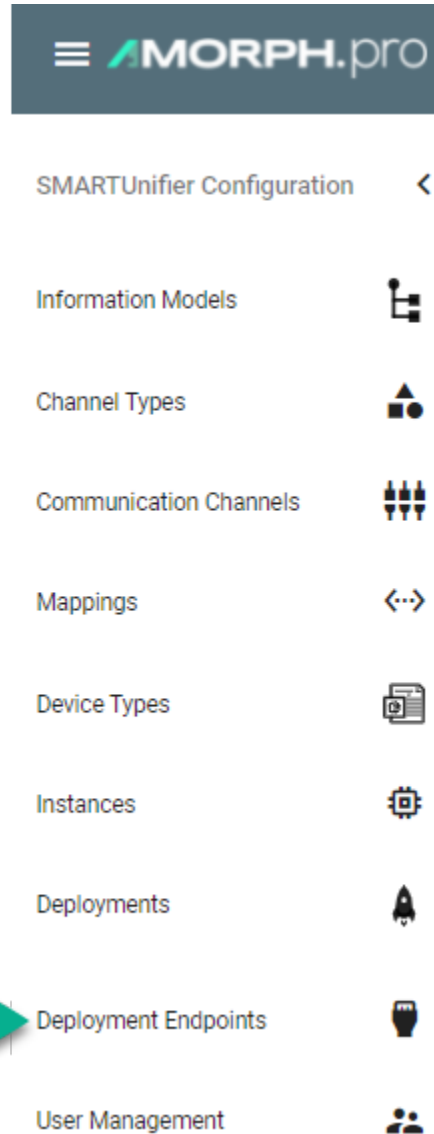
Deployment Endpoints are used to identify the location of a Deployment (i.e., the definition where an Instance is executed). With the Deployment Endpoints, you can create and maintain those locations. This feature can only be accessed by a user with the administrator role.

#### 4.1.2 Deployment Endpoints Types

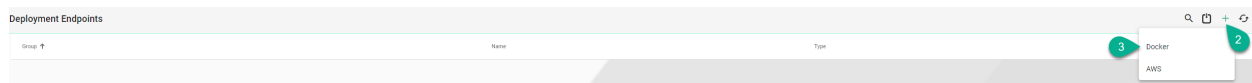
##### Docker

SMARTUNIFIER supports the Deployment of Instances using Docker Containers. Before creating a new Deployment for an Instance using Docker, install Docker on your device and open up the [Docker Remote API Interface](#). If you want to learn more about Docker and how to install it, visit the [Docker Website](#). When your Docker Daemon is up and running you have to provide a Docker endpoint.

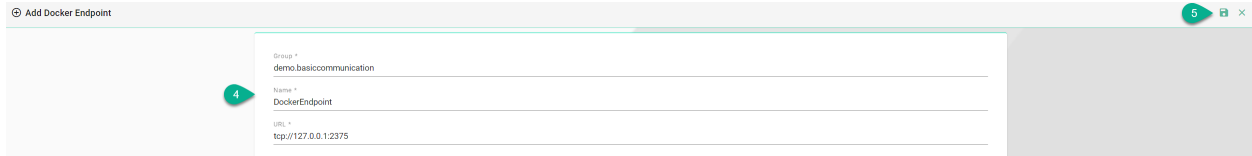
- Navigate to the SMARTUNIFIER Deployment Endpoints perspective (1).



- Click on the “Add Endpoint” button (2).
- Select the Deployment Type **Docker** from the pop-up (3).



- In the “Add Endpoint” view a set of configuration parameters is required (4)
  - Provide a **Group** and a **Name**
  - Provide **URL**. Depending on your use case choose between the **unix** e.g., `unix:///var/run/docker.sock` or the **tcp** e.g., `tcp://127.0.0.1:2375` **protocol**.
- After all mandatory fields are filled in, click the “Save” button (5).

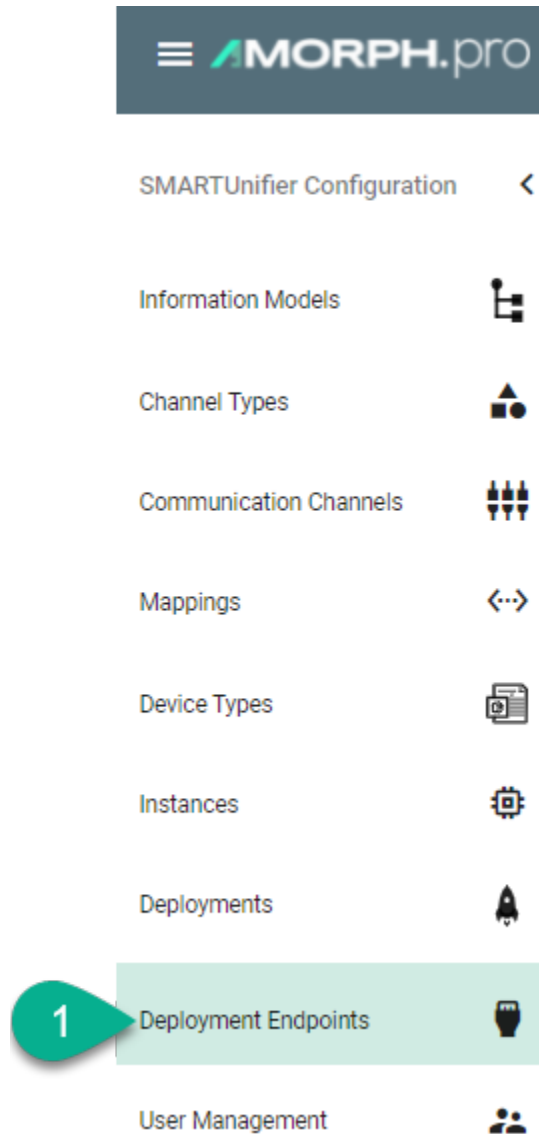


## AWS

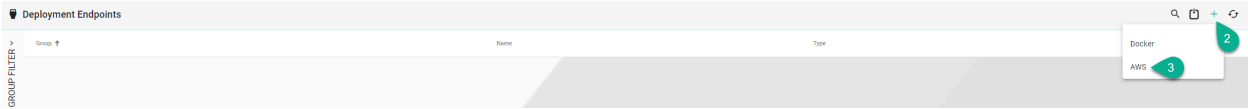
Before deploying a SMARTUNIFIER Instance on AWS Fargate you need to create an AWS Deployment Endpoint. The AWS Deployment Endpoint specifies, which AWS account should be used for the deployment.

Follow the steps described below to create an AWS Deployment Endpoint:

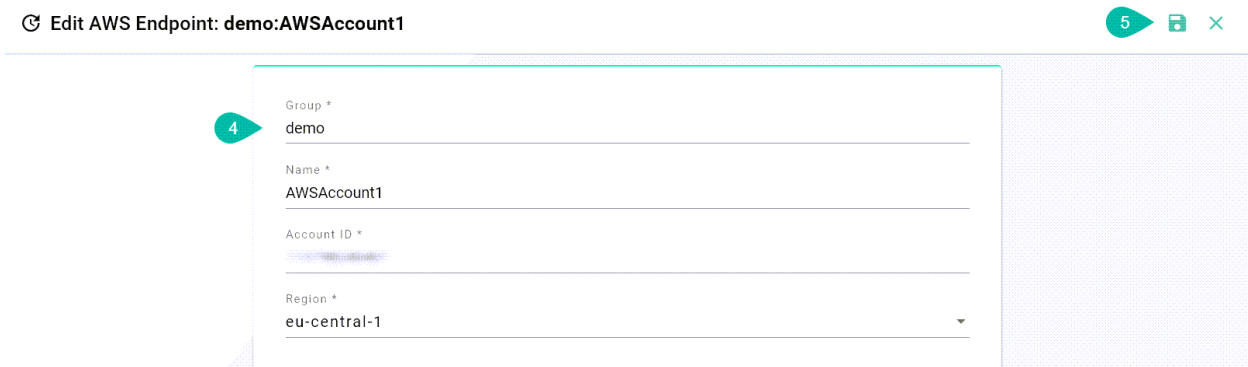
- Select the SMARTUNIFIER Deployment Endpoints perspective (1).



- Click the “Add” button (2).
- Select AWS (3).



- Configure your AWS account by entering the following parameters (4):
  - Enter a Group and a Name.
  - Enter your AWS account ID.
  - Select the [region](#).
- Save the new Endpoint by clicking the “Safe” button (5):



## 4.2 User Management

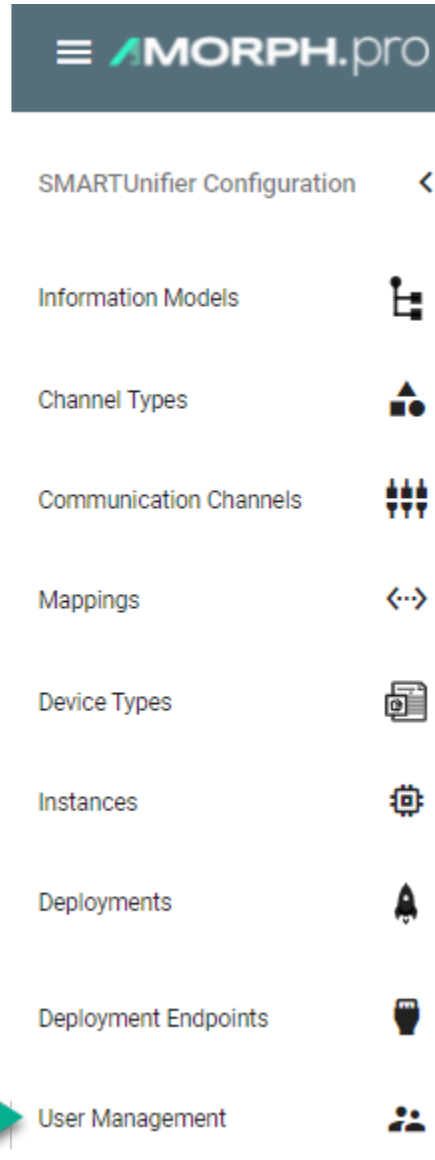
### 4.2.1 About User Management

Within the User Management the administrator can create users accounts, assign permissions as well as activate or deactivate user accounts.

### 4.2.2 Add a new user

This procedure describes how to create a new user account.

- Select the SMARTUNIFIER User Management perspective (1).



- Click the “Add User” button (2).

**User Management** 2 + ↻

User ID	Email	First Name	Last Name	Language	Role	Status	Created	
admin		Unifier	Admin	en	Administrator	Active	2020-07-13 00:00:00.000	 

- In the “Add User” view you have to provide the following information (3):
  - Provide a **user id, first and last name**
  - Optionally, provide an e-mail address
  - Set a preferred language for the SMARTUNIFIER Manager.
- The role defines the permission of the user. It is mandatory to assign a new user a role. The following roles are available for use in the SMARTUNIFIER.

- **Administrator:** Full read and write access for the SMARTUNIFIER Configuration and Administration.
- **Reader:** Only read access for the SMARTUNIFIER Configuration
- **Writer:** Read and write access for the SMARTUNIFIER Configuration
- Choose the account status: Active or Inactive.
  - **Active:** User account is activated and ready to use.
  - **Inactive:** User account is deactivated and cannot be used until it is activated again.
- Set an initial password for the first login of the new user.
- After all mandatory fields are filled in, click the “Save” button (4).

The screenshot shows the 'Add User' form with the following fields and values:

- User ID: JohnDoe2 (marked with callout 3)
- Email: (empty)
- First Name: John
- Last Name: Doe
- Language: English
- Role:
  - Administrator:
  - Reader:
  - Writer:
- Status:
  - Active:
  - Inactive:
- Credentials:
  - Password: (masked with dots)
  - Confirm password: (masked with dots)

Below the password fields, there are four password requirements:

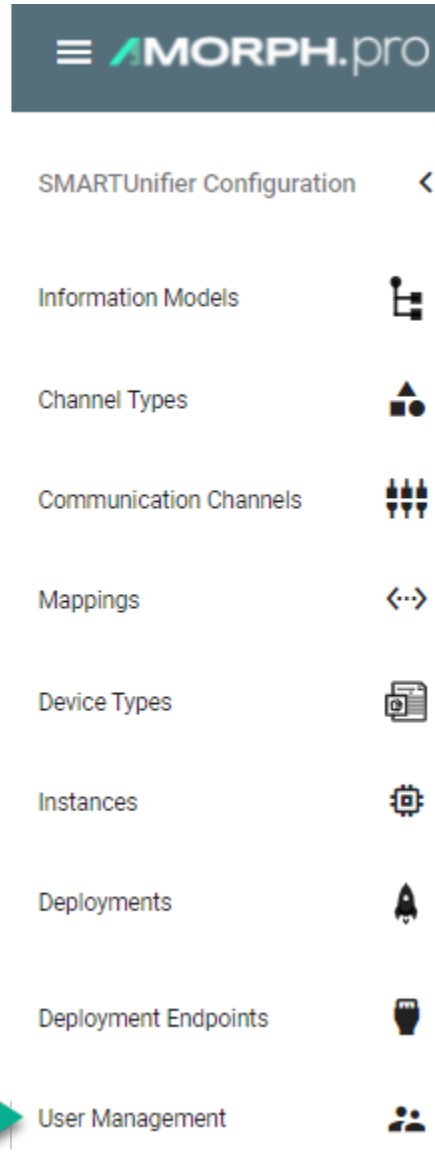
- ✓ contains at least one lower character
- ✓ contains at least one upper character
- ✓ contains at least one digit character
- ✓ contains at least one special character
- ✓ contains at least 4 characters

A green callout '4' points to the Save button in the top right corner of the form.

### 4.2.3 Edit a user

This procedure describes how to edit an existing user account.

- Select the SMARTUNIFIER User Management perspective (1).



- Click the “Edit” button (2).

User Management								Search	+	Refresh
User ID ↑	Email	First Name	Last Name	Language	Role	Status	Created			
JohnDoe2		John	Doe	en	Reader	Active	2021-03-26 00:00:00.000	2	✎	☰
admin		Unifier	Administrator	en	Administrator	Active	2021-03-26 00:00:00.000		✎	☰

In the “Edit” view the user account can be redefined (3).

- update the user details: user id, first and last name, email address
- change the language
- edit the user permission: Administrator, Writer or Reader
- **activate** or **inactivate** the user account



- change the password

3

4

Edit User: John Doe

User ID \*  
JohnDoe2

Email

First Name \*  
John

Last Name \*  
Doe

Language \*  
English

Role: Reader

Administrator

Reader

Writer

Status: Active

Active

Inactive

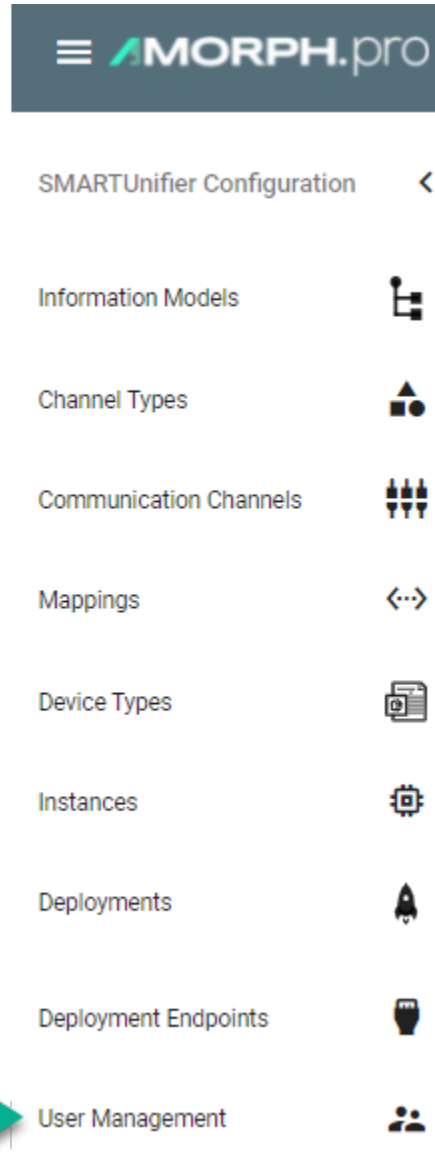
Change Password

- After editing, click the “Save” button (4).

#### 4.2.4 Delete a user

This procedure describes how to delete a user account.

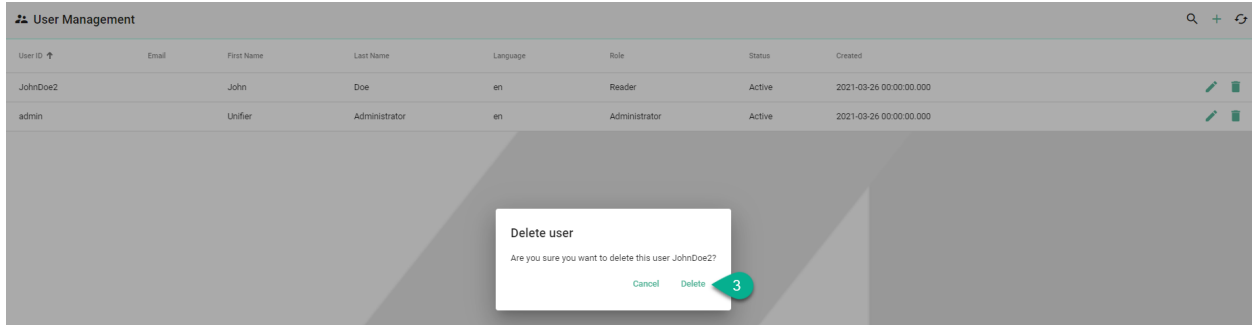
- Select the SMARTUNIFIER User Management perspective (1).



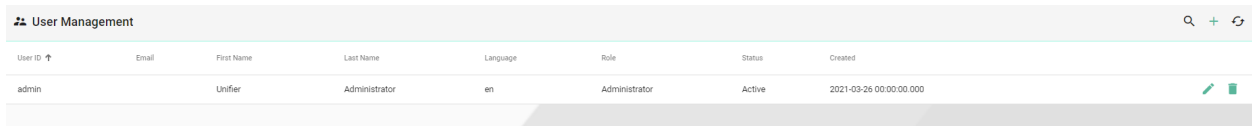
- Click the “Delete” button (2).

User Management								🔍	+	↻
User ID ↑	Email	First Name	Last Name	Language	Role	Status	Created			
JohnDoe2		John	Doe	en	Reader	Active	2021-03-26 00:00:00.000			2
admin		Unifer	Administrator	en	Administrator	Active	2021-03-26 00:00:00.000			

Confirm by selecting the “Delete” button (3).



The user account is deleted and no more visible in the SMARTUNIFIER User Management perspective.



## DEMONSTRATION SCENARIOS

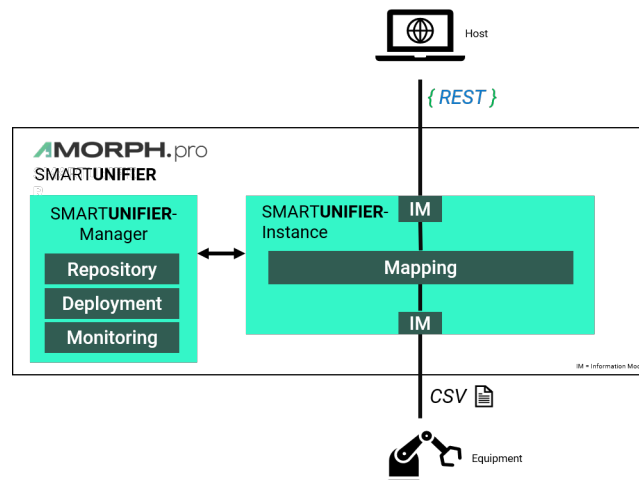
You want to learn and get your hands on a demonstration integration scenario? Check out the following sample demonstration uses cases:

- *CSV file data to REST Server*
- *Insert JSON data in SQL-Database*
- *XML file data to MQTT with database operation*

### 5.1 File-based data - CSV to REST-Server

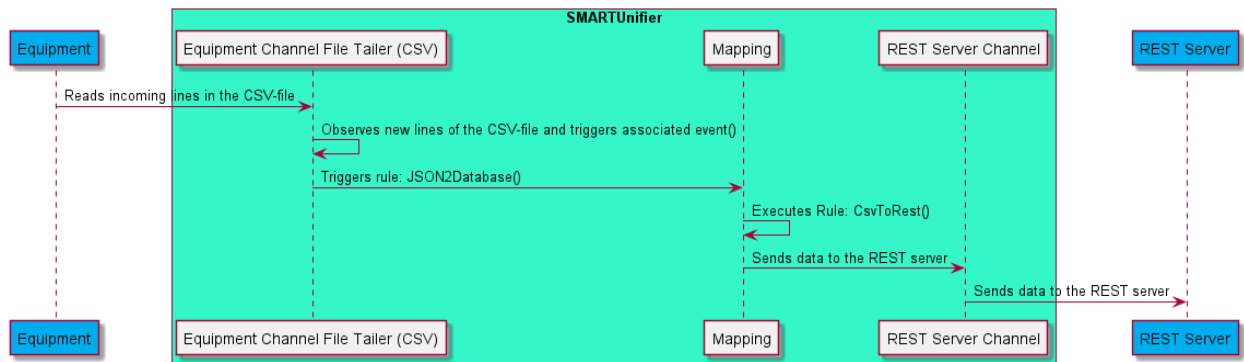
#### 5.1.1 Overview

This Scenario describes step by step how an integration of equipment data to any kind of REST-server is done. The next steps guide you through the creation of Information Models, Channels, Mappings, Device Types, Instances and Deployments.



The CSV, which represents the equipment data, in this demo scenario contains four parameters that are all comma-delimited. Below you can find the sample data. Create a new CSV-file on your local machine and copy and paste the sample data.

```
"PARTNR", "TIMESTAMP", "TEMPERATUR", "PRESSURE"
"4595", "2020-05-01 07:00:43", "62", "222"
"4596", "2019-05-01 07:01:43", "62", "223"
"4597", "2019-05-01 07:02:43", "63", "223"
"4598", "2019-05-01 07:03:43", "61", "225"
"4599", "2019-05-01 07:04:43", "66", "228"
"4600", "2019-05-01 07:05:43", "64", "223"
"4601", "2019-05-01 07:06:43", "66", "223"
"4602", "2019-05-01 07:07:43", "62", "222"
"4603", "2019-05-01 07:08:43", "62", "228"
```



## 5.1.2 Information Model

### CSV-file

The first step is to create an Information Model that represents the structure of your CSV-file.

What Information Models are and how to create one is described in chapter *Information Models*.

You can see the Information Model for the data of the CSV-file below. The Model “CsvDataModel” has an Event “csvDemoEvent”. Inside the Event you find the same parameter as in the CSV-file.

We recommend to use the same “Group” name throughout the scenario. For example, in order to identify the created Artifacts for this scenario the group name “demo.basiccommunication” is used in this documentation.

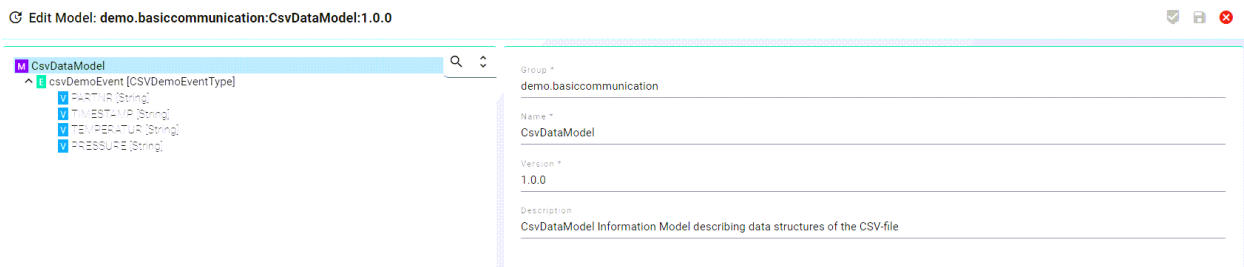


Table 1: CsvDataModel - Variables

ID	Node Type	Data Type
csvDemoEvent	Event	CSVDemoEvent
PARTNR	Variable	String
TIMESTAMP	Variable	String
TEMPERATUR	Variable	String
PRESSURE	Variable	String

## REST-Server

There has to be also a second Information Model for the REST-Server.

The “RestDataModel” has a structured Variable called “RestDemoData”, which holds the variables “Temperatur” and “Pressure”.

Since only these two values are sent from the CSV-file to the Rest Server, it is not necessary to add the other parameters of the CSV-file.

You can see the Information Model in the screenshot below as well as the values used in the table.

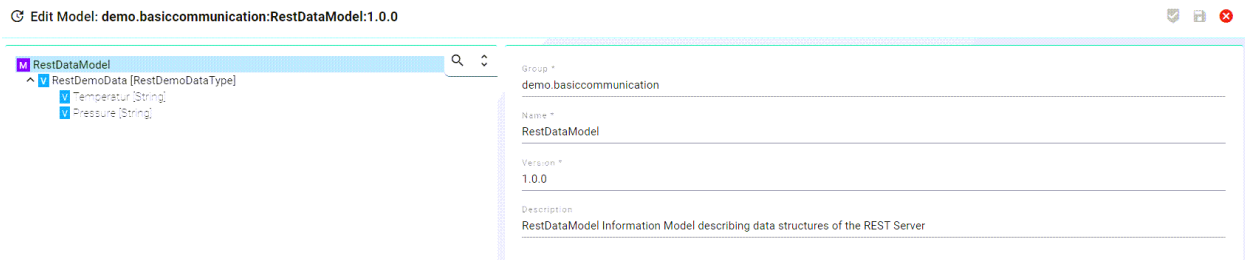


Table 2: RestDataModel - Variables

ID	Definition Type	Member Type
RestDemoData	Variable	RestDemoDataType
Temperatur	Variable	String
Pressure	Variable	String

### 5.1.3 Communication Channel

#### File Tailer

Next step is to, create a Communication Channel for the CSV-file.

- 1.) Enter values for group, name and version like in the screenshot below.
- 2.) Select the model for the CSV-file as the Information Model connected to this Channel.
- 3.) Select “**File tailer (CSV)**” as the Channel Type.

⊞ Add Communication Channel

Group \*  
demo.basiccommunication

Name \*  
CSVChannel

Version \*  
1.0.0

Description

Model \*  
demo.basiccommunication.CsvDataModel.1.0.0

Channel type \*  
File tailer (CSV)

## REST-Server

Similar to the CsvDataModel, create a Channel for the RestDataModel.

- 1.) Enter values for group, name and version like in the screenshot below.
- 2.) Select the Model “**demo.basiccommunication:RestDataModel:1.0.0**”.
- 3.) Lastly, select as a Channel Type the “**RestServer**” Channel.

⊞ Edit Communication Channel: demo.basiccommunication:RESTServerChannel:1.0.0

Group \*  
demo.basiccommunication

Name \*  
RESTServerChannel

Version \*  
1.0.0

Description

Model \*  
demo.basiccommunication.RestDataModel.1.0.0

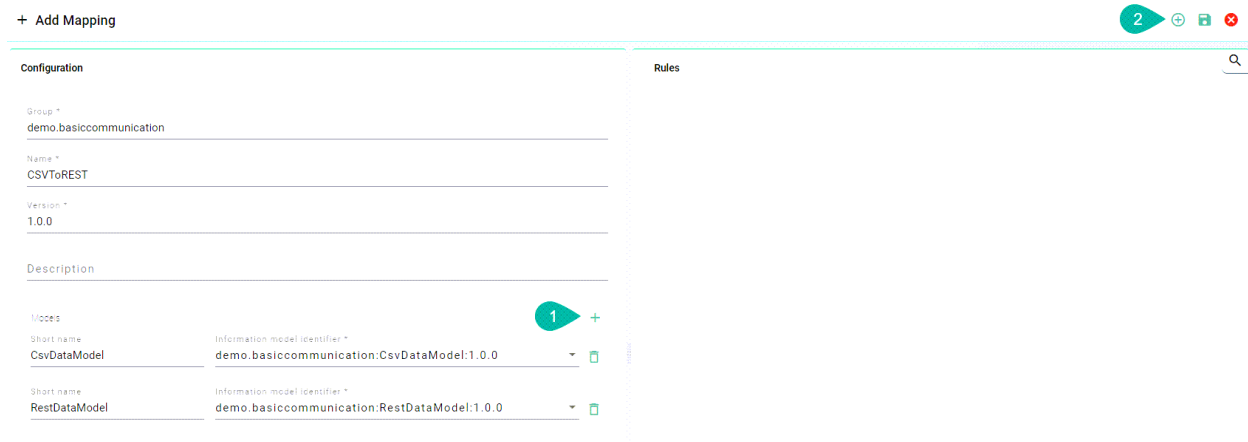
Channel type \*  
REST Server

The configuration of the CSV-Channel, as well as the REST-Channel, is done in the section *Create Instance*.

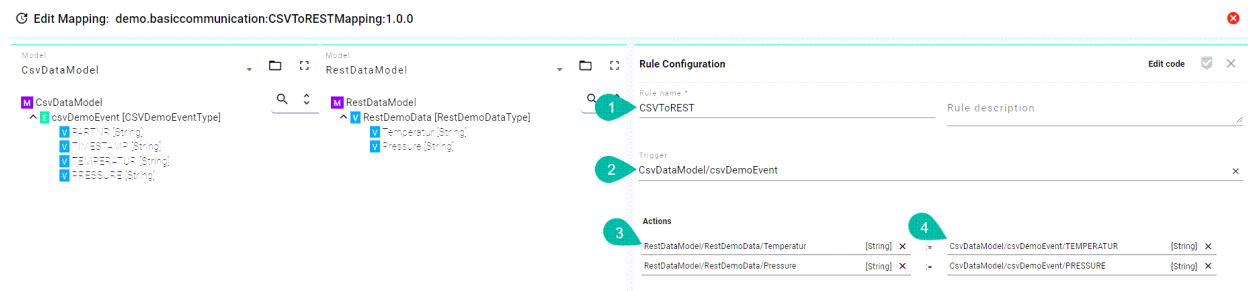
### 5.1.4 Mapping

After the creation of Information Models and their dependent Communication Channels create in the next step the Mapping.

- 1.) Enter values for Group, Name and Version like in the screenshot below.
- 2.) Select the Information Models created earlier. Click the “Add” button (1) and select the Information Model “**demo.basiccommunication:CsvDataModel:1.0.0**”. Enter a Name for the Information Model, e.g., “**CsvDataModel**”.
- 3.) Click again the “Add” button (1) and select the Information Model “**demo.basiccommunication:RestDataModel:1.0.0**”. Enter a Name, e.g., “**RestDataModel**”.
- 4.) Next, add a Rule to the Mapping. Click the “Add Rule” button (2).



- 5.) Enter a Name for the new Rule (1) like “CsvToRest”.
- 6.) As Trigger drag and drop the “**csvDemoEvent**” of the CsvDataModel into the Trigger field (2). This Event is going to be triggered if any changes appear inside the CSV-file.
- 7.) Drag and Drop the “**Temperature**” and the “**Pressure**” variable from the “**RestDataModel**” as a new Target (3)
- 8.) Drag and Drop the belonging variables “**TEMPERATURE**” and “**PRESSURE**” from the “**CsvDataModel**” into the Source fields (4)



### 5.1.5 Device Type

Next, assign the Mapping to a new Device Type.

- 1.) Enter values for Group, Name and Version like in the Screenshot below.
- 2.) Click the “Add Mapping” button (1).
- 3.) Select the Mapping “**demo.basiccommunication:CSVToREST:1.0.0**” previously created (2).
- 4.) Assign the correct Channels to the Information Models (3).
- 5.) Save the new Device Type.



Edit Device Type: demo.basiccommunication:SUDeviceType:1.0.0

Group \*  
demo.basiccommunication

Name \*  
SUDeviceType

Version \*  
1.0.0

Description

Mappings + 1

2 Mapping  
demo.basiccommunication:CSVToRESTMMapping:1.0.0

Models	Channels
CsvDataModel	demo.basiccommunication:CSVChannel:1.0.0 3
RestDataModel	demo.basiccommunication:RESTServerChannel:1.0.0 3

## 5.1.6 Instance

Last step of this Scenario is the creation of the Instance.

- 1.) Select the Device Type “**demo.basiccommunication:CSVDemoDT:1.0.0**” previously created.
- 2.) Values for Group, Name and Version are already set from the Device Type. Although, we recommend to change the Name.

Add Instance

1 Device Type  
demo.basiccommunication:SUDeviceType:1.0.0

Group \*  
demo.basiccommunication

Name \*  
SUDeviceType

Version \*  
1.0.0

Description

Mappings

Mapping  
demo.basiccommunication:CSVToRESTMMapping:1.0.0

Models	Channels
CsvDataModel	demo.basiccommunication:CSVChannel:1.0.0 2
RestDataModel	demo.basiccommunication:RESTServerChannel:1.0.0 3

- 3.) Click the configuration button for the “**demo.basiccommunication:CSVChannel:1.0.0**” configuration (2).

3.1) Set the “**file path**” for the location of the CSV-file on your device.

3.2) Enter a value in milliseconds for the “**delay between checks**” of the CSV-file for new content.

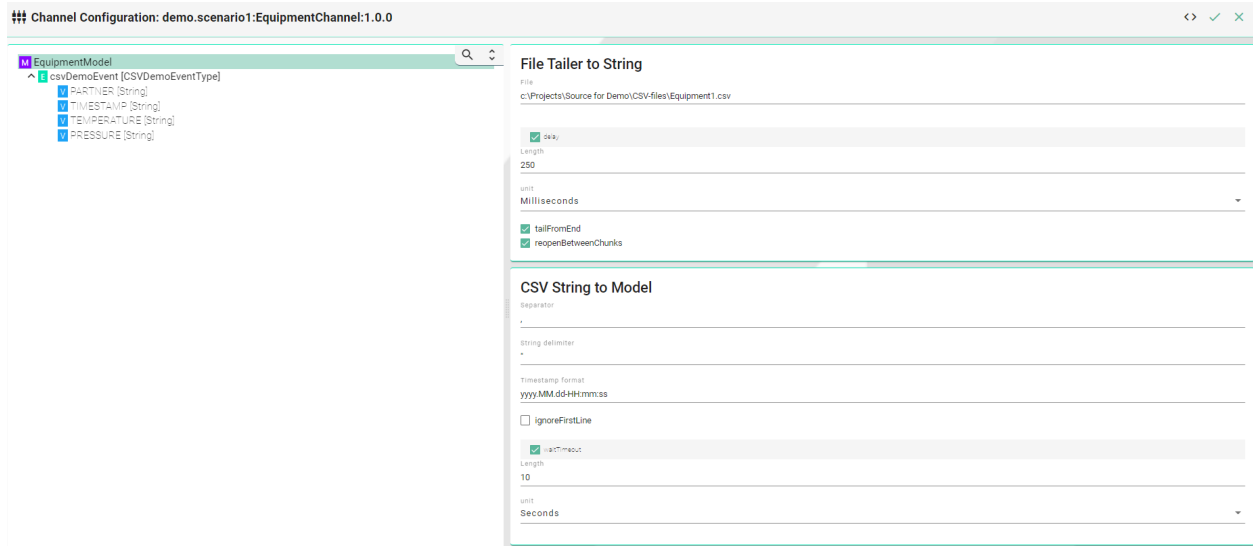
3.3) In order to tail from the end set “**tailFromEnd**” to true.

3.4) Set “**reopenBetweenChunks**” to true to close and reopen the file between reading chunks. In this example it is set to false.

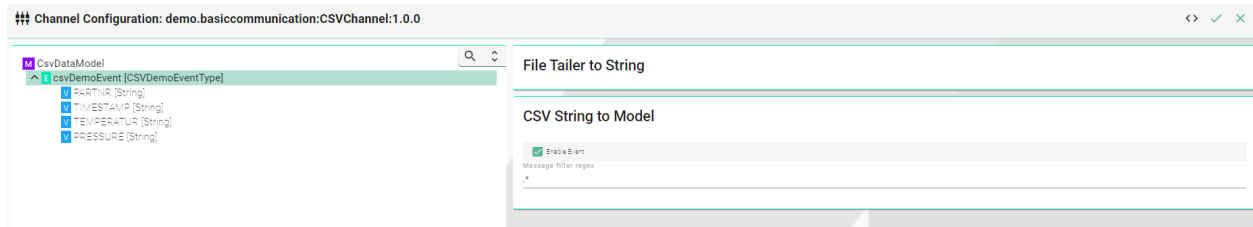
3.5) Since the CSV-file is comma-delimited enter , as separator

3.6) Since the values in the CSV-file each start and end with double quotation marks, enter “ as String Delimiter.

3.7) Since the Timestamp values are not used in the CSV-file, the use of a Timestamp format is not necessary.



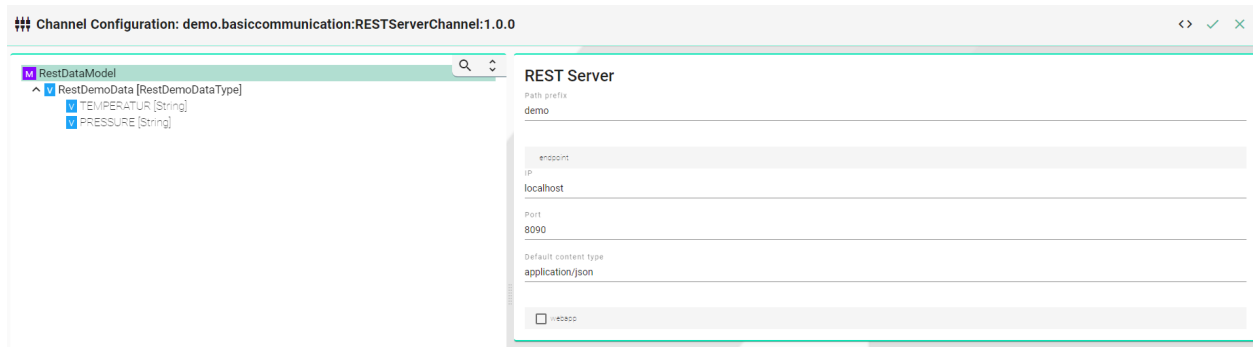
3.8) Select the “**csvDemoEvent**” node in the Information Model on the left hand side. Enter “.\*” as the message filter RegEx.



4.) Lastly, it follows the configuration of the Rest Server Channel

4.1) Enter a value for the path prefix like “**demo**”.

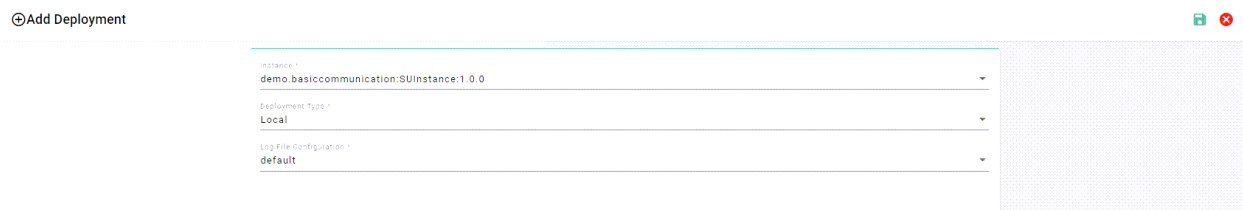
4.2) Leave the default settings for the Rest Server Endpoint as it is. We recommend to change the “**Port**” if the default is occupied.



After all steps have been executed the Instance “**instance.demo.basiccommunication:DemoDTCSV:1.0.0**” is fully configured and ready for Deployment.

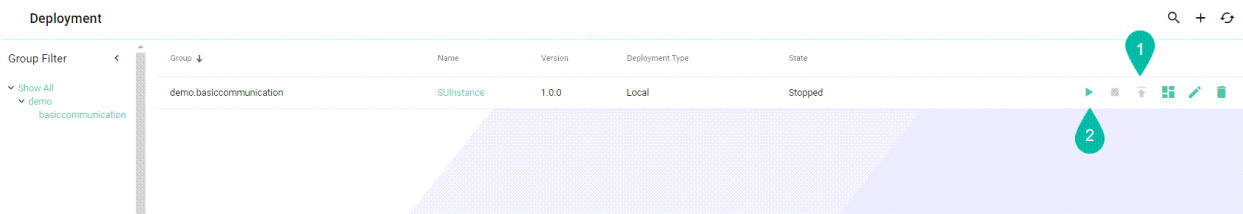
### 5.1.7 Deployment

- 1.) Select the Instance “**instance.demo.basiccommunication:DemoDTCSV:1.0.0**” previously created.
- 2.) Select “**Local**” as Deployment Type. This deploys the Instance on the machine you are working on.
- 3.) Leave “**LogFileConfiguration**” on “**default**”.



- 4.) Click the “Deploy” button (1)

- 5.) Click the “Start” button (2)



The Instance is now running on your local machine. You can now see the data inside a Browser. Therefore you can either show both values or only one value.

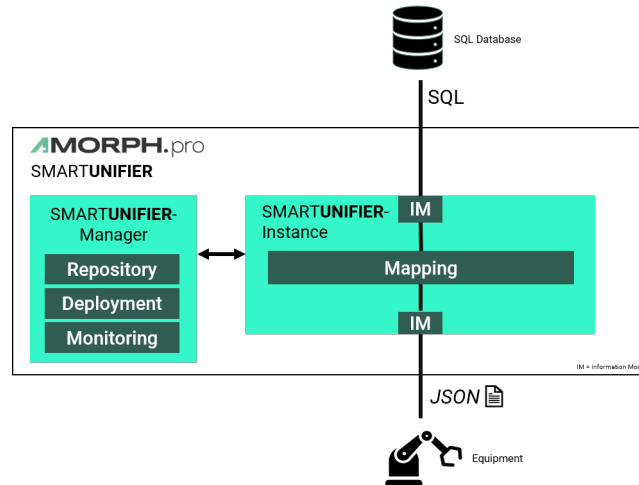
- Open any Browser and enter <http://localhost:8091/demo/Variable/RestDemoData> as an URL to receive the latest values for both Pressure as well Temperature.
- Open any Browser and enter <http://localhost:8091/demo/Variable/RestDemoData/Variable/Temperatur> as an URL to get the latest value for the value Temperature.
- Open any Browser and enter <http://localhost:8091/demo/Variable/RestDemoData/Variable/Pressure> as an URL to get the latest value for the value Pressure.

**Warning:** Please note that the URL must match the naming of the prefix set in the configuration (Step 5.1) as well as the naming of variables in the Information Model for the REST-Server. E.g., If the custom variable *RestDemoData* is changed to *RestDemo* the url must be changed accordingly: `<http://localhost:8091/demo/Variable/RestDemoData>`.

## 5.2 File-based data - Insert JSON data in SQL-Database

### 5.2.1 Overview

This Scenario describes step by step how JSON-data can be inserted into an SQL database with the SMARTUNIFIER.



### 5.2.2 Prerequisite

#### JSON-file

```
{
  "equipmentId": "Equipment_A1234",
  "orderNr": "Order_000101",
  "materialNr": "A1C55100",
  "quality": "IO"
}
```

#### SQL-Server Database

Create Database Command: CREATE DATABASE unifier

Create Schema Command: CREATE SCHEMA dbo

Create Table Command: create table dbo.SU\_DEMO\_UC1\_TABLE(EQUIPMENT\_ID varchar(max), ORDER\_NR varchar(max), MATERIAL\_NR varchar(max), QUALITY varchar(max))

Or any other SQL Database supported by SMARTUNIFIER.

### 5.2.3 Information Model

#### JSON-file

Create an Information Model that represents the structure of the JSON-file

Structure of the *JSONFile* Information Model:

- Event that represents the trigger for the Mapping
- Variables under the Event represent the key-value pairs from the JSON-file

🔄 Edit Model: demo.basiccommunication.uc1:JSONFile:1.0.0

The screenshot displays the configuration for the **JSONFile** Information Model. On the left, a tree view shows the model structure: **JSONFile** (parent) contains **FileEvent [FileEventType]** (child), which in turn contains four variables: **equipmentId [String]**, **orderId [String]**, **materialNr [String]**, and **quantity [String]**. On the right, the configuration form is filled with the following details:

- Group \***: demo.basiccommunication.uc1
- Name \***: JSONFile
- Version \***: 1.0.0
- Description**: (empty field)

## SQL-Database

Create an Information Model that represents the database

Structure of the *Database* Information Model:

- Event that represents the table of the database
- Variables under the Event represent the columns within the table

🔄 Edit Model: demo.basiccommunication.uc1:Database:1.0.0

The screenshot displays the configuration for the **Database** Information Model. On the left, a tree view shows the model structure: **Database** (parent) contains **DatabaseInsertEvent [DatabaseInsertEventType]** (child), which in turn contains four variables: **EquipmentId [String]**, **orderId [String]**, **MaterialNr [String]**, and **Quantity [String]**. On the right, the configuration form is filled with the following details:

- Group \***: demo.basiccommunication.uc1
- Name \***: Database
- Version \***: 1.0.0
- Description**: (empty field)

## 5.2.4 Communication Channel

### JSON-file

In this scenario the JSON-file is processed by the SMARTUNIFIER with the build-in File Consumer

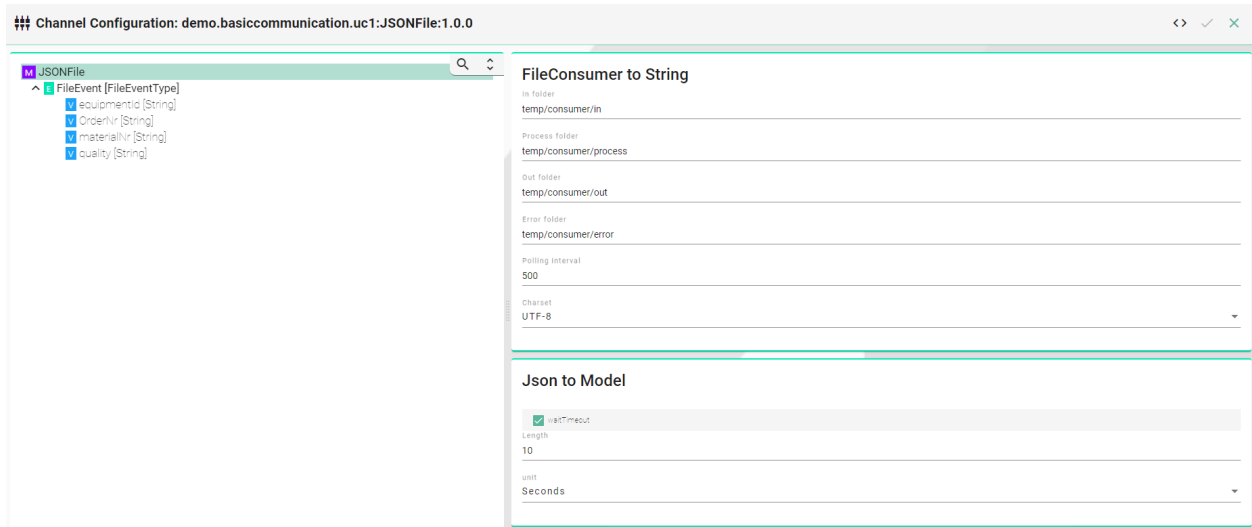
Create File Consumer Channel:

- Select the *JSONFile* Information Model created previously
- Select *File reader (JSON)* as Channel Type

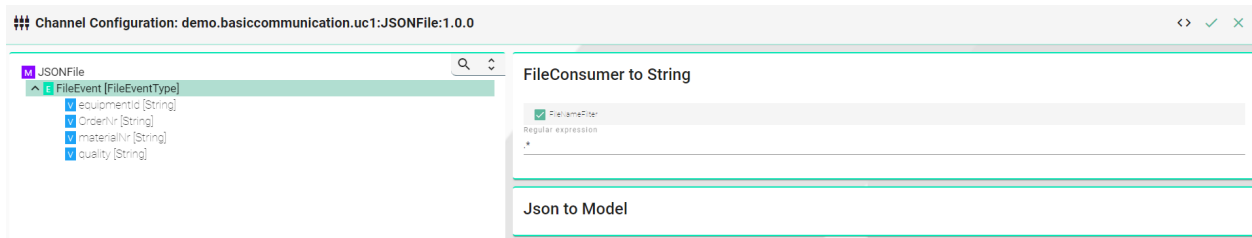
Configuration:

- Specify paths to following folder:

- InFolder
- ProcessFolder
- OutFolder
- ErrorFolder



- Select the Event to configure the *FileNameFilter*



## SQL Database

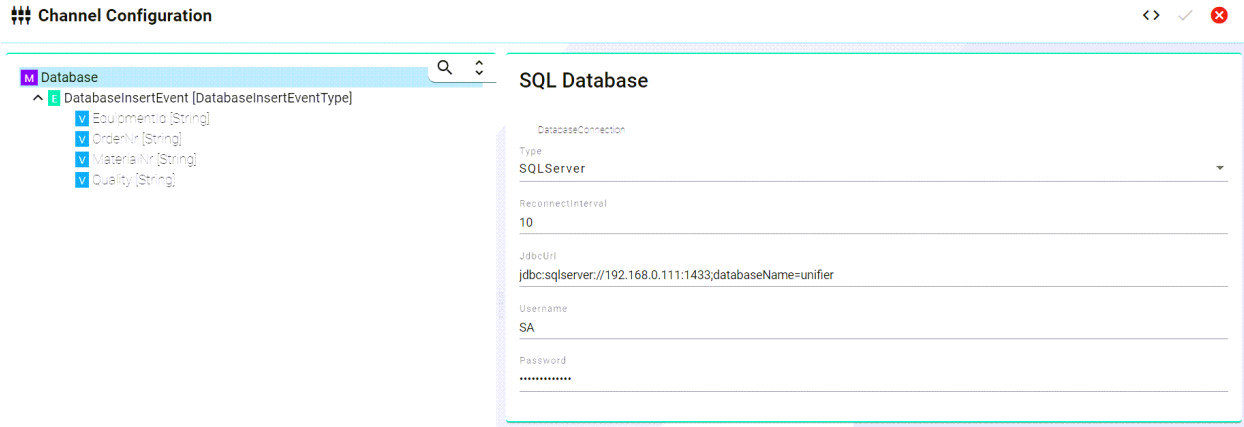
In this scenario the JSON-file is processed by the SMARTUNIFIER with the build-in File Consumer.

Create Database Channel:

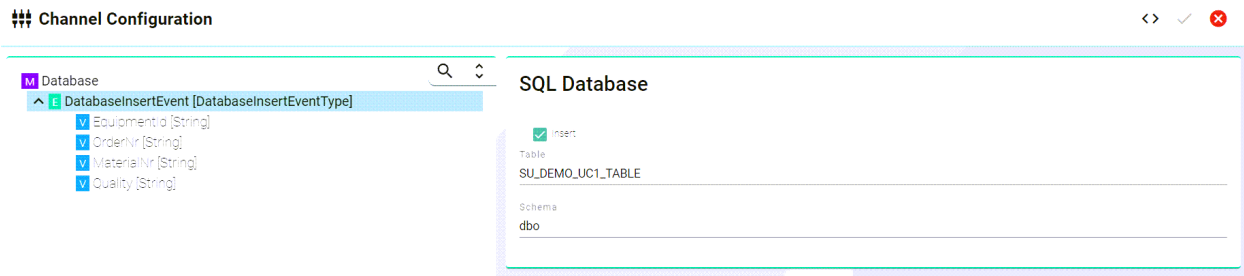
- Select the *Database* Information Model created previously
- Select *SqlDatabase* as Channel Type

Configuration:

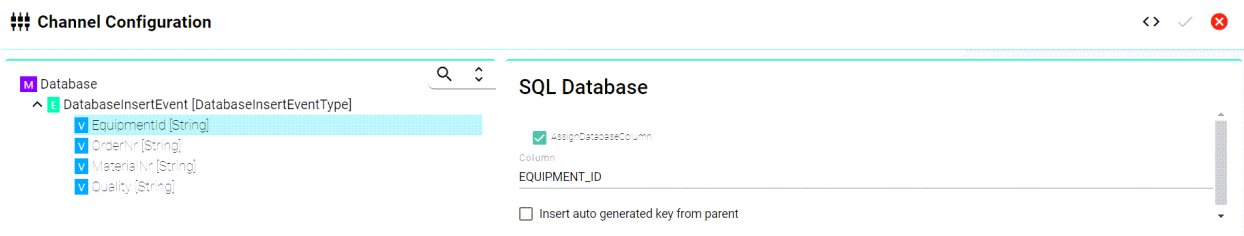
- Select the root node of the Information Model and configure the database access
  - Select *SQLServer* as Type
  - Enter the *JdbcUrl* e.g., `jdbc:sqlserver://127.0.0.1:1433;databaseName=unifier`
  - Enter username and password of the database



- Select the Event node in the Information Model and configure the table settings
  - Enable the checkbox *Insert*
  - Enter the name for the *Table* as well as the *Schema*



- Select the Variable node *EquipmentId* in the Information Model and configure the columns (Repeat this step with the rest of the Variables)
  - Enable the checkbox *AssignDatabaseColumn*
  - Enter the name for the *Column*



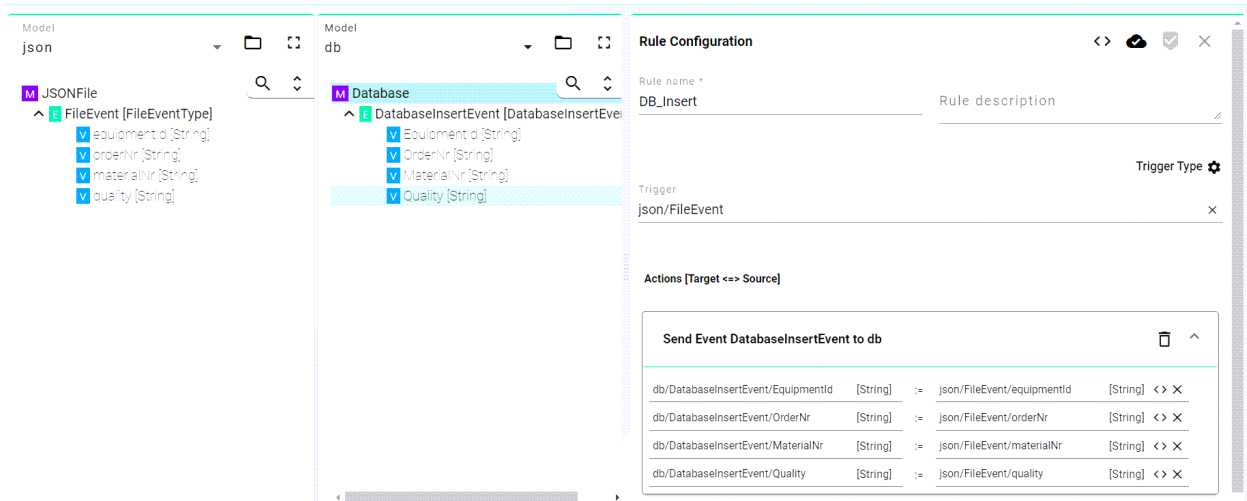
## 5.2.5 Mapping

Create a new Mapping with the Information Models created previously (JSONFile and Database). Create a Rule that handles the assignment of values from the JSON-file to the database.

- Enter a *Rule Name*
- Drag and Drop the *File Event* into the Trigger field

- Drag and Drop the *DatabaseInsertEvent* into the Actions panel
- Assign source to target (Repeat for all Variables)
- Drag and Drop the Variable *equipmentId* from the json model into the according Source field

Edit Mapping: demo.basiccommunication.uc1:JsonToDB:1.0.0



## 5.2.6 Device Type

Create a new Device Type.

- Select the Mapping *JsonToDB* created previously
- Assign the Channels (Database and JSONFile) to their belonging Information Model

Edit Device Type: demo.basiccommunication.uc1:UC1\_DeviceType:1.0.0



## 5.2.7 Instance

Create a new Instance



- Select the Device Type *UC1\_DeviceType* created previously
- Change the name of the Instance to *UC1\_Instance*
- If necessary: Changes of the Channel configuration can be made

Edit Instance: demo.basiccommunication.uc1:UC1\_Instance:1.0.0



Group \*  
demo.basiccommunication.uc1

Name \*  
UC1\_Instance

Version \*  
1.0.0

Description

Mappings

Mapping  
demo.basiccommunication.uc1:JsonToDB:1.0.0

Models	Channels
db	demo.basiccommunication.uc1:Database:1.0.0
json	demo.basiccommunication.uc1:JSONFile:1.0.0

## 5.2.8 Deployment

Create a new **Local** Deployment

- Select the Instance *UC1\_Instance*
- Select the Log File Configuration *Info* (Defines the log level)

Edit Deployment: demo.basiccommunication.uc1:UC1\_Instance:1.0.0



Instance  
demo.basiccommunication.uc1:UC1\_Instance:1.0.0

Log File Configuration \*  
Debug

Deploy and Start the Instance *UC1\_Instance*

Deployment

Group Filter < Group Name Version Deployment Type State

Group Filter	Group	Name	Version	Deployment Type	State
<ul style="list-style-type: none"> <li>Show All</li> <li>demo               <ul style="list-style-type: none"> <li>basiccommunicati_                   <ul style="list-style-type: none"> <li>uc1</li> </ul> </li> </ul> </li> </ul>	demo.basiccommunication.uc1	UC1_Instance	1.0.0	Local	Started

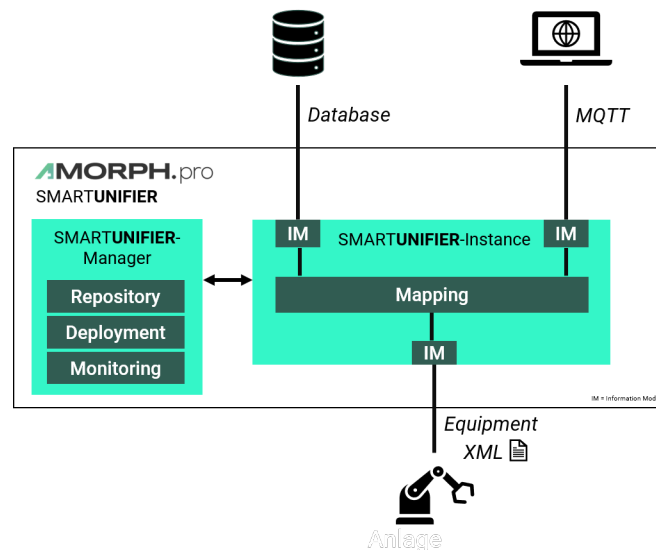
## Execution

In order to insert the data of the JSON-file into the SQL database move the file into the specified **InFolder**.

## 5.3 File-based data - XML, Database, and MQTT

### 5.3.1 Overview

This Scenario describes step by step how XML data can be send via MQTT enriched with additional data from a database. This scenario shows also how type conversion and date formatting can be implemented via the SMARTUNIFIER Mapping.



### 5.3.2 Prerequisites

#### 1. Equipment Data - (XML file)

```
<?xml version="1.0" encoding="utf-8"?>
<ProductionResult>
  <OrderNumber>PO_000001</OrderNumber>
  <ProductNumber>F2PZJ55QW11</ProductNumber>
  <Date>2021-03-31T07:20:41.214Z</Date>
  <Quality>I0</Quality>
  <Quantity>5</Quantity>
</ProductionResult>
```

#### 2. SQL Server (Database)

##### Create Table

```
create table DEMO_INTEGRATION_UC3_SCHEMA.CUSTOMER (MAIN_KEY bigint IDENTITY(1,1)
PRIMARY KEY, CUSTOMER_NAME nvarchar(max), ORDER_NUMBER nvarchar(max))
```

##### Insert Data

```
INSERT INTO DEMO_INTEGRATION_UC3_SCHEMA.CUSTOMER (CUSTOMER_NAME, ORDER_NUMBER)
VALUES ('DemoCompany1', 'PO_000001'); INSERT INTO DEMO_INTEGRATION_UC3_SCHEMA.
CUSTOMER (CUSTOMER_NAME, ORDER_NUMBER) VALUES ('DemoCompany1', 'PO_000002');
```

```
INSERT INTO DEMO_INTEGRATION_UC3_SCHEMA.CUSTOMER (CUSTOMER_NAME, ORDER_NUMBER)
VALUES ('DemoCompany2', 'PO_000003'); INSERT INTO DEMO_INTEGRATION_UC3_SCHEMA.CUSTOMER
(CUSTOMER_NAME, ORDER_NUMBER) VALUES ('DemoCompany3', 'PO_000004');
```

### 3. MQTT Client (For testing)

Download the [MQTT Explorer](#).

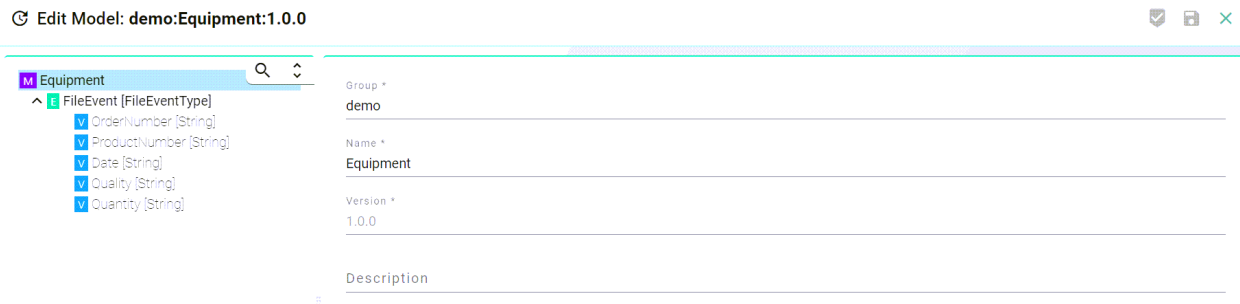
## 5.3.3 Information Model

### Equipment

Create an Information Model that represents the structure of the XML-file.

Structure of the *XML* - Information Model:

- Event that represents the **trigger** for the Mapping. If a new file is recognized by SMARTUNIFIER the Rule in the Mapping will be executed.
- Variables (of data type String) under the Event represent the key-value pairs from the XML-file

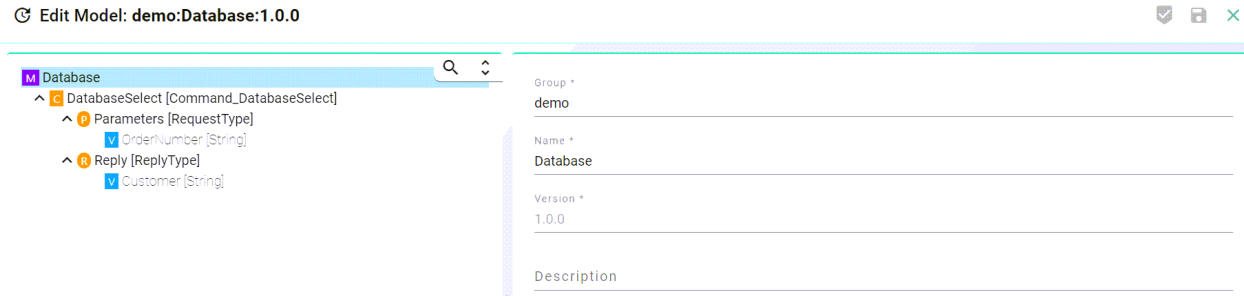


### Database

Create an Information Model, which will be used for the **Select** query.

Structure of the *Database* - Information Model:

- Command which is executed once the trigger is activated.
- Parameter variable **OrderNumber** (of data type String) that is used in the SELECT query later on.
- Reply variable **Customer** (of data type String) that holds the result of the query.

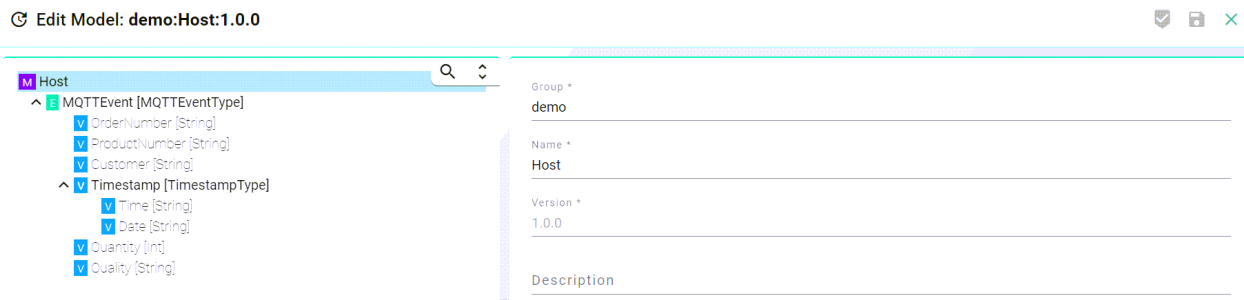


## Host (MQTT)

Create an Information Model that represents the structure of the Host (in this case MQTT).

Structure of the *MQTT* - Information Model:

- Event which is used to trigger the transfer of the data.
- Variables:
  - OrderNumber, ProductNumber, Customer, Quality - of type String.
  - Quantity of type Int
  - Timestamp (custom data type)
    - \* date, time - of type String



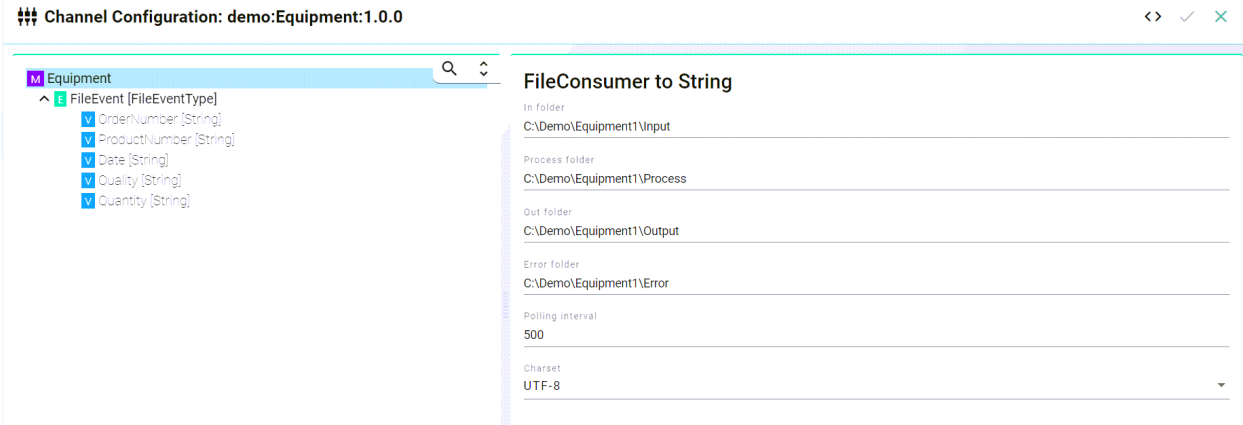
## 5.3.4 Communication Channel

### Equipment

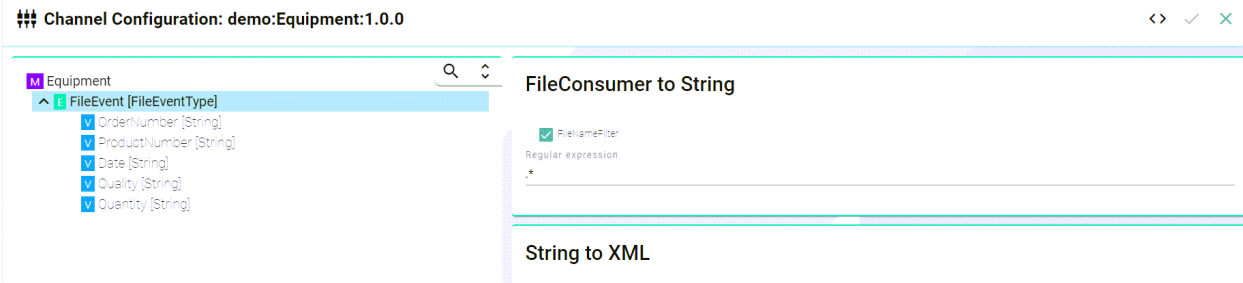
In this scenario the XML-file is processed by the SMARTUNIFIER with the build-in File Reader.

1. Create File Reader Channel:
  - Select the **Equipemnt** Information Model created previously.
  - Select **File Reader (XML)** as Channel Type.
2. Configuration:
  - Specify paths to following folder:

- InFolder
- ProcessFolder
- OutFolder
- ErrorFolder

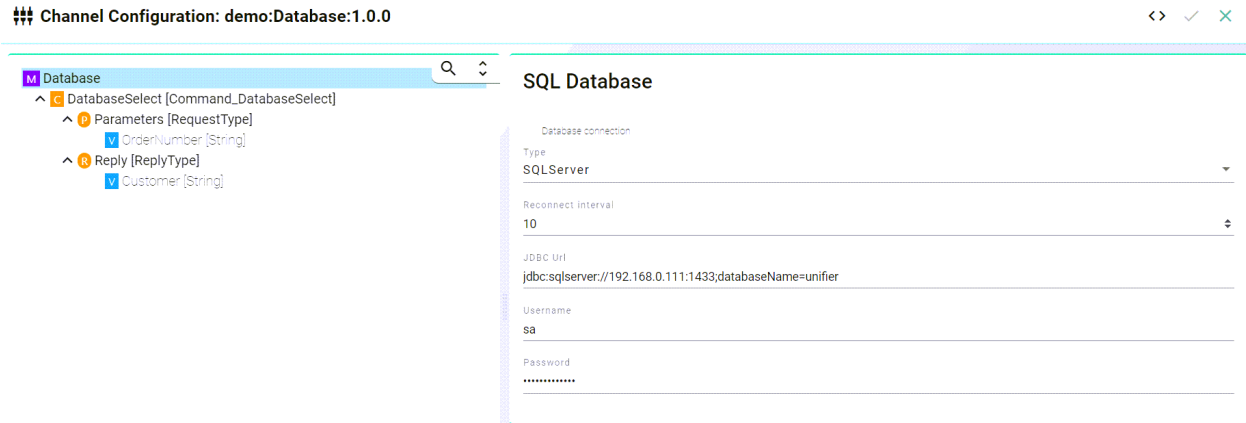


- Select the Event to configure the *FileNameFilter*

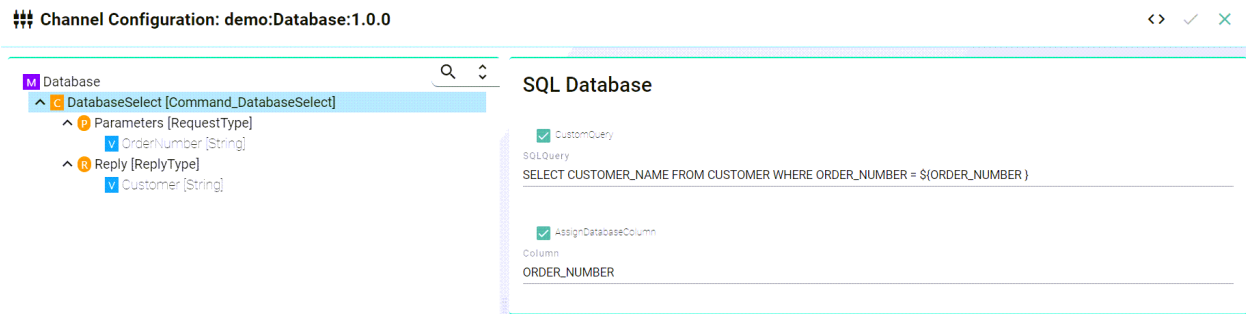


## SQL Database

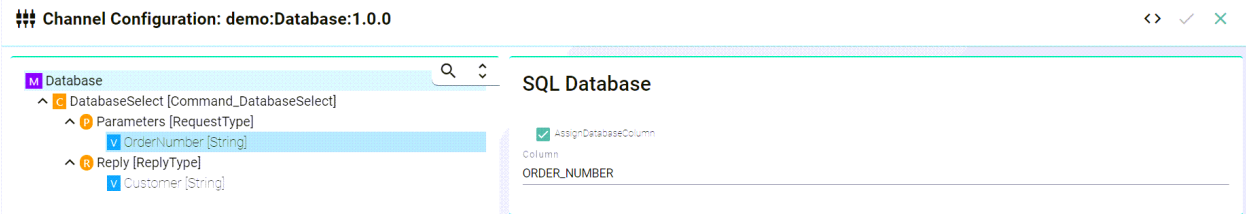
1. Create a SQL Database Channel:
  - Select the **Database** Information Model created previously.
  - Select **SqlDatabase** as Channel Type.
2. Configuration:
  - Database Connection:
    - Select the database type **SQLServer**.
    - Set the **JDBC Url** - jdbc:sqlserver://192.168.0.111:1433;databaseName=unifier
    - Enter **username** and **password**.



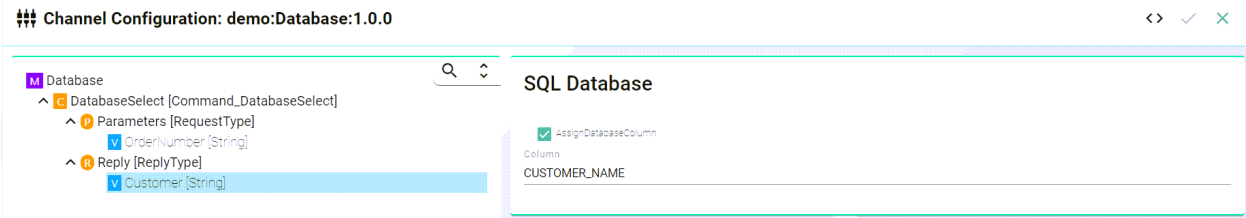
- Enter the SELECT query - select CUSTOMER\_NAME from DEMO\_INTEGRATION\_UC3\_SCHEMA.CUSTOMER where ORDER\_NUMBER = \${ORDER\_NUMBER}



- Enter the database column for the parameter **OrderNumber**.



- Enter the database column for the result variable **Customer**.



## MQTT

1. Create the MQTT Channel:
  - Select the **Host** Information Model created previously.

- Select **MQTT (Json)** as Channel Type.
2. Configuration:
- Enter the **IP** of the MQTT Client.
  - Enter the **Port** of the MQTT Client.

Channel Configuration: demo:Host:1.0.0

MQTT to String

Host: 127.0.0.1

Port: 1884

Requested Interval: 5

Connection Timeout: 60

Keep-alive Interval: 60

Persistence Folder: persist/\$(ClientId)

Client ID: 001

Retained

Username:

Password:

HostnameVerification

TraceToS

DisconnectedBuffer

- Select the Event to enable the checkbox **Producers** and enter a **topic** name.

Channel Configuration: demo:Host:1.0.0

MQTT to String

Consumers

Producers

Topic: demo/order\_details

Json to Model

### 5.3.5 Mappings

Create a Mapping with the Information Models created previously (Equipment, Database, and Host).

Create a Rule that executes a the SELECT query and sends the result with the other equipment data out via MQTT.

- Enter a **Rule Name**
- Select the **Edit Code** button.

**Note:** This scenario does not support the drag-and-drop functionality of the SMARTUNIFIER

Mapping due to type conversion and date formatting.

- Use the following code snippet and paste it into the Rule code editor:

```
equipment.FileEvent mapTo { event =>
  database.DatabaseSelect.execute(command => {
    command.OrderNumber := event.orderNumber
    CommunicationLogger.log(event, command)
  }, reply => {
    host.MQTTEvent.send(event1 => {

      event1.Timestamp.time := java.time.format.DateTimeFormatter.ofPattern("HH:mm:ss").
      ↪format(java.time.OffsetDateTime.parse(event.date.value.toString))
      event1.Timestamp.date := java.time.format.DateTimeFormatter.ofPattern("dd.MM.yyyy").
      ↪format(java.time.OffsetDateTime.parse(event.date.value.toString))

      event1.OrderNumber := event.orderNumber
      event1.ProductNumber := event.productNumber
      event1.Customer := reply.Customer
      event1.Quantity := event.quantity.toInt
      event1.Quality := event.quality

      CommunicationLogger.log(reply, event1)
    }
  )
}
}
```

Edit Mapping: demo:OrderInformationToMQTT:1.0.0

The screenshot displays the SmartUnifier interface for editing a mapping. On the left, two model trees are visible: 'db' and 'file'. The 'db' model contains a 'Database' entity with a 'DatabaseSelect' component (parameters: OrderNumber, Reply, Customer) and a 'Parameters' component (parameters: OrderNumber, Reply, Customer). The 'file' model contains an 'Equipment' entity with a 'FileEvent' component (parameters: OrderNumber, ProductNumber, Date, Quality, Quantity). On the right, the 'Rule Configuration' pane shows a rule named 'DataToMQTT' with a rule description that contains the code snippet from the previous section.

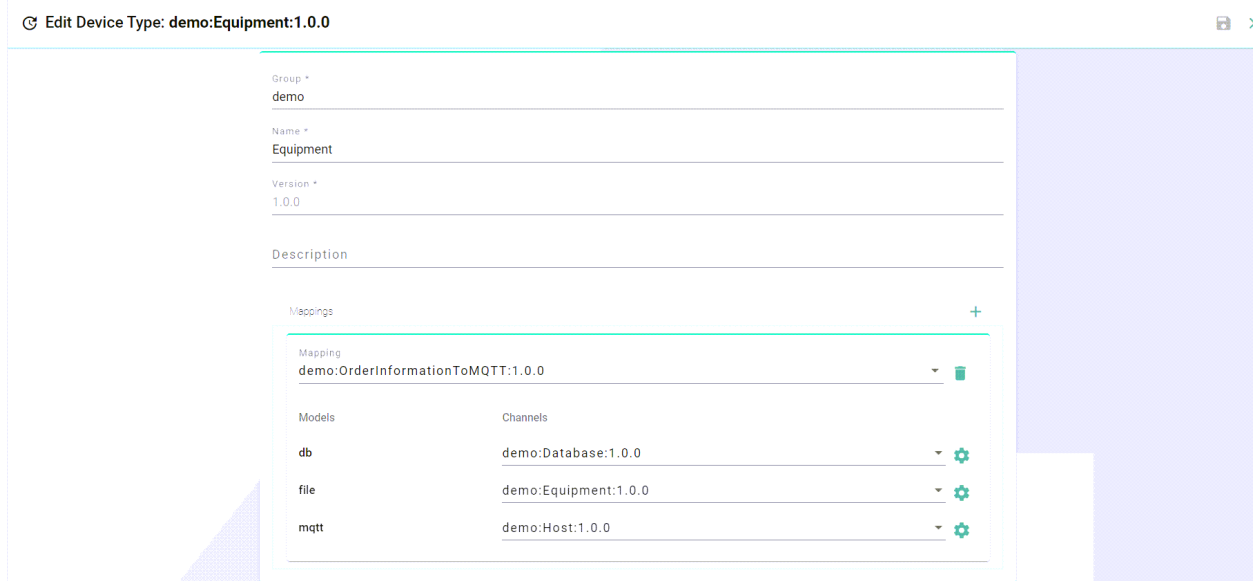
### 5.3.6 Device Type

Create a Device Type.

- Select the Mapping **EquipmentToHost** created previously



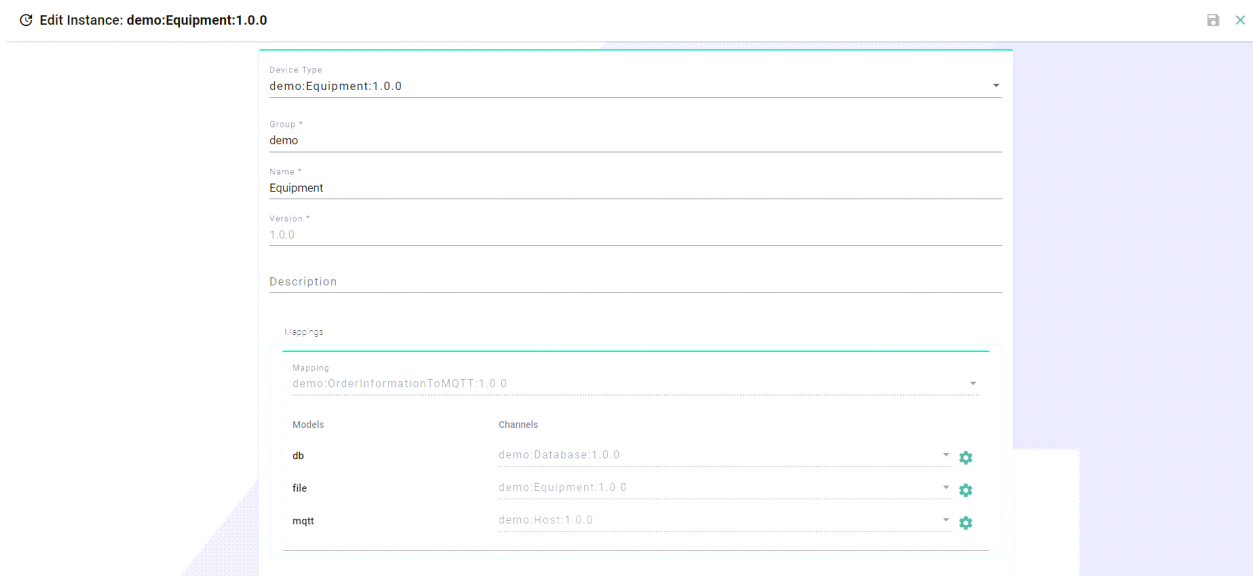
- Assign the Channels (Equipment, Database and Host (MQTT)) to their belonging Information Models.



### 5.3.7 Instance

#### Create a Instance

- Select the *Device Type* created previously.

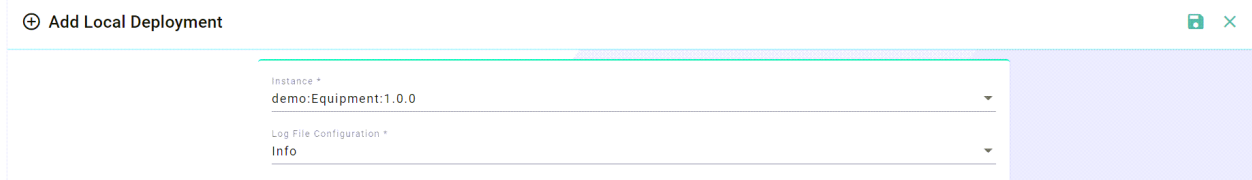


### 5.3.8 Deployment

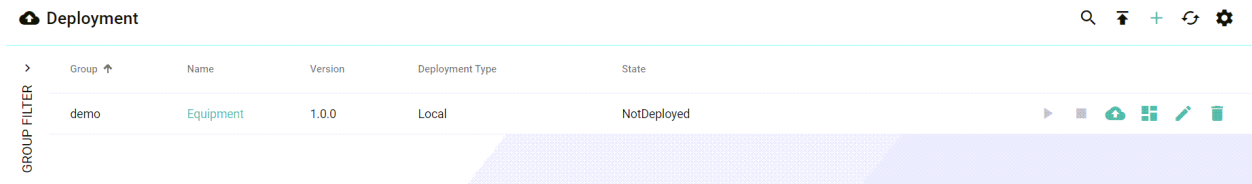
#### Create a new Local Deployment

- Select the *Instance* created previously.

- Select the Log File Configuration *Info* (This defines the log detail).



### Deploy and Start the Instance *UC1\_Instance*



### Execution

In order to send the data from the equipment with the customer information via MQTT, move the XML-file into the specified **InFolder**.

## GETTING HELP

Having trouble? We'd like to help!

- In case of malfunctioning SU Instances check out the *Troubleshooting* section.
- Try the *FAQ* - it's got answers to regularly asked questions.
- Check out the Glossary if some terminology is not clear.

### 6.1 Troubleshooting

#### 6.1.1 Instance works abnormally or hangs/crashes

1. Determine if it is a HW problem on the Hardware where the SU Instance is operated on.
2. In case of a HW problem setup a new HW (or switch to a spare HW). Ensure to place correct security certificates on the new HW. Perform new deployment and start of a new SU Instance with SU Manager on the new HW.
  - Time to redeploy and start SU Instance: **< 2 min.**

**In case, the HW is operating correctly, perform the following steps:**

1. Verify if SU Instance is still running or has crashed.
2. Save SU Instance logs and provide logs to Amorph Systems support for further analysis. Then stop and restart the SU Instance with SU Manager. See if SU Instance is up and running again correctly. Otherwise perform next steps.
  - Time to restart SU Instance: **< 30 seconds.**
3. Stop and perform undeploy of SU instance with SU Manager; Afterwards again deploy the SU Instance and start the SU Instance again. See if SU Instance is running correctly. Otherwise perform next steps.
  - Time to redeploy and start SU Instance: **< 2 min.**
4. If at SU Instance's side everything works correctly but still no correct communication takes place, go through the next steps.

5. Check all attached connection channels; see if these are running correctly by verifying SU logs and verifying, if correct data is received or sent by the SU Instance. In case the SU Instance sends data correctly to receivers (see SU log entries), but still communication is not working, perform incident measures on receivers' side. **Make sure** that the receivers are working correctly. In case SU receives no data from a sender (see SU log entries), perform incident measure on sender's side. **Make sure**, the sender is providing its data in a correct way.
6. In case there is still no correct communication, contact Amorph Systems' support for further assistance.

### 6.1.2 Manager works abnormally or hangs/crashes

1. Determine if it is a HW problem on the HW where SU Manager is operated.
2. In case of a HW problem, setup a new HW or switch to a spare HW. Perform installation of SU Manager and Repository from latest backup and re-start SU Manager on the new HW.
  - Time to install and start SU Manager: < 3 min.

**In case HW is operating correctly, perform the following steps:**

1. Verify that SU Manager is still running or has crashed.
2. Save SU Manager logs and provide logs to Amorph Systems' support for further Analysis. Then stop and restart the SU Manager. See if SU Manager is running correctly. Otherwise perform next steps.
  - Time to restart SU Manager: < 30 seconds
3. Perform complete uninstall of SU Manager and Repository; Perform new installation of SU Manager and Repository from latest backup and re-start SU Manager.
  - Time to re-install and start SU Manager: < 3 min.
4. In case there is still no correct operation of SU Manager, perform previous step with an older backup that has proven to work correctly.
5. In case there is still no correct operation of SU Manager, contact Amorph Systems support for further assistance.

## 6.2 FAQ

**Does SMARTUNIFIER provide caching/buffering of data?**

Yes, SMARTUNIFIER is capable of supporting caching of messages using file buffer (Spool) for message transfer to external middleware like MQTT. This functionality can be provided as part of a SMARTUNIFIER Communication Channel and dependent on the used communication protocol of the respective channel.

**Is it possible to set different buffering options for different channels?**

Yes, each communication channel of SMARTUNIFIER can provide a different buffer size and further options.

**Does SMARTUNIFIER enable data pre-processing, cleansing, filtering and optimization of data?**

Yes, this is a core feature of SMARTUNIFIER. SMARTUNIFIER provides powerful capabilities for any kind data preprocessing, cleansing, filtering and optimization. The capabilities of SMARTUNIFIER in this respect range from simple calculations, unit conversions, type conversions and reformatting up to arbitrary processing algorithms of any complexity.

**Does SMARTUNIFIER enable data aggregation?**

Yes, SMARTUNIFIER enables data aggregation and reformatting with any level of complexity.

**Does SMARTUNIFIER provide short term data historian features?**

Yes, historic telemetric data (of variable time horizons; size limited by used HW) can be monitored by usage of SMARTUNIFIER's logs which can record all communication activities of a SMARTUNIFIER Instance incl. telemetric data. SMARTUNIFIER's Log data can afterwards be forwarded by usage of a dedicated Communication Channel to any (and also multiple) upper-level monitoring or analytics system. Alternatively SMARTUNIFIER's Logs can be accessed directly by any external IT application (remote access to HW device is required).

Yes, SMARTUNIFIER can create any number of OPC-UA Servers and/or Clients within just one Communication Instance.

**Does SMARTUNIFIER support standard number of connections to OPC-UA Clients?**

Yes, SMARTUNIFIER supports a virtually unlimited number of client connections per OPC-UA Server. Physically the number of connections is limited by number of subscriptions per session, number of data objects and size per subscription as well as by HW and network constraints. SMARTUNIFIER allows to operate multiple OPC-UA Servers and/or OPC-UA Clients within each single SMARTUNIFIER instance for northbound and/or southbound communication.

**Does SMARTUNIFIER support brokering to MQTT Server?**

Yes, SMARTUNIFIER supports any number of MQTT connections. One single SMARTUNIFIER Instance can connect to one or multiple MQTT brokers (e.g., for different target systems) and is able to communicate bi-directional.

### Which southbound protocols are offered with SMARTUNIFIER?

SMARTUNIFIER supports many protocols like e.g.,

- Siemens S7, S7-2
- OPC-UA
- Beckhoff
- MQTT
- Modbus-TCP
- file-based (different formats like CSV, XML, JSON, any binary format)
- SQL

... and many more to come continuously. Specific protocols can be provided based on customer request. Therefore please contact Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com)).

### Does SMARTUNIFIER enable pre-aggregation of additional sensor data and/or more devices (rule based), for e.g., temperature monitoring?

Yes, SMARTUNIFIER allows to connect any number of telemetric data sources to a SMARTUNIFIER Instance. Rule-based pre-aggregation and pre-processing of additional sensor data is supported with any level of complexity. This ranges from simple pre aggregation/pre-processing up to complex utilization of advanced AI or ML algorithms.

### Does SMARTUNIFIER support processing of active cloud commands? (e.g., System Manager AWS / AWS Agent)

Yes, SMARTUNIFIER provides a RESTful API to execute Shell Commands (e.g., Start/Stop Instance, etc.). Thus, active cloud commands are supported. In addition, also commands from other external IT-Systems (e.g., MES, ERP, AWS Systems Manager etc.) are possible. Furthermore if required SMARTUNIFIER can be fully executed and operated within Cloud Environments (e.g., within AWS Cloud).

### Which northbound protocols are supported by SMARTUNIFIER?

SMARTUNIFIER supports many northbound protocols, like e.g.,

- OPC-UA
- MQTT
- WebSphere
- HTTP / REST
- any file based protocol
- SQL/any database

- Splunk
- Vantiq

...and many more to come continuously. Specific protocols can be provided based on customer request. Therefore, please contact Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com)).

### **Does SMARTUNIFIER support international naming standards (example: EUROMAP 77, PackML)?**

Yes, SMARTUNIFIER is specifically designed to support strongly the incorporation of international standards (e.g., EUROMAP 77, 82, 83, 84, AutomationML, PackML, DFQ, SEMI SECS/GEM etc.) as well as company standards, by offering the capability to be able to build up specific SMARTUNIFIER Information Models complying with these standards and incorporating full data semantics. There will be a one-time effort to implement such a standard in SMARTUNIFIER as a respective Information Model and afterwards this Standard can be used for any communication across the whole customer IT Infrastructure. Also this includes flexible mapping from legacy protocols to new standard protocol and vice versa.

### **Does SMARTUNIFIER offer the ability to integrate with other systems and applications through REST Server APIs and Web Services for Operational purpose?**

Yes, SMARTUNIFIER features a REST API for operational purpose (e.g., instance start/stop service, configuration etc.)

### **Does SMARTUNIFIER offer a way to realize a flexible, configurable dataflow?**

Yes, SMARTUNIFIER features a configurable and highly performant rule-based engine (SmartMappings) based on different northbound and/or southbound input sources for realizing any dataflow (workflow) that is required in industrial environments. This covers communication sequences for identification, processing start, processing execution, processing end, results data provision as well as detailed process data provision. Also commands from any upper-level IT-System can be processed and further transmitted to the production equipment (e.g., recipe management, NC program transfer etc.) External data flow engines / visualization apps (e.g., Node-Red, Grafana) can be connected.

### **Does SMARTUNIFIER enable Central Software Management?**

Yes, all Information Models, Mappings and Deployment Features can be managed centrally. Furthermore, SMARTUNIFIER features an easy to use REST API for operational purpose (e.g., instance start/stop service, configuration etc.).

### **Does SMARTUNIFIER enable Container Deployment?**

Yes, SMARTUNIFIER operation and deployment is fully based on Container-Technology (Docker). SMARTUNIFIER Manager and Instances can be operated and deployed inside Docker Containers to any End Point within the network running Docker environment.

### **Which Operating System SMARTUNIFIER is supporting?**

SMARTUNIFIER runs on Windows, Linux, Mac and other OS supporting Java RT and Docker.

### **Does SMARTUNIFIER support onPrem Edge-Analytics?**

Yes, SMARTUNIFIER can be connected to any Edge-Analytics System SMARTUNIFIER Logs can provide detailed information about all communication activities. These log data can either be provided by a dedicated Communication Channel to any upper level Analytics System (in any required format) or can be made locally accessible to any agent running locally on the HW.

### **Does SMARTUNIFIER support DevOps CI/CD Pipeline for installations and update?**

Yes, SMARTUNIFIER supports remote installation/update of Software from SMARTUNIFIER Manager via Docker Registry SMARTUNIFIER Instances (running in Docker Containers) can be updated, monitored and controlled remotely. Docker registry is also accessible from external systems if required.

### **Does SMARTUNIFIER enable Software Scalability?**

Yes, SMARTUNIFIER is able to scale from connection of one single equipment/device to virtually any number of equipment/devices by means of its decentralized architecture.

### **Does SMARTUNIFIER support the architecture of distributed systems?**

Yes, SMARTUNIFIER itself is a fully distributed and scalable IT system. With this architecture SMARTUNIFIER is able to collaborate in any small or large IT environment. SMARTUNIFIER is open to reliably collaborate in large sites.

### **Does SMARTUNIFIER provide the ability to directly communicate with other Devices or IT-Systems through standard protocols and also supports Load-Balancing?**

Yes, SMARTUNIFIER is able to communicate with any other Devices or IT-Systems and also address load balancers for optimized feeding of data to any message brokers or data forwarder.

### **Does SMARTUNIFIER provide the ability for data to be ingested as a consolidated batch (File Transfer)?**

Yes, SMARTUNIFIER is capable of using any files in any formats as input source and also as output destination.



**Does SMARTUNIFIER provide the ability to create custom connectors to ingest data from arbitrary sources?**

Yes, the capability to be able to realize custom connectors for any data source is one of the core elements of SMARTUNIFIER's architecture.

**Is SMARTUNIFIER able to push operational data to an Edge-Gateway?**

Yes, SMARTUNIFIER is capable of receiving operational data from any device or IT-System and push it to an Edge-GW. E.g., OPC-UA, MQTT and HTTP/REST are supported. Also, many other protocols can be used therefore.

**Does SMARTUNIFIER provide Software Monitoring?**

Yes, each SMARTUNIFIER Instance creates detailed logs that document every communication activity. These logs can be made accessible to any external system e.g., by a dedicated monitoring communication channel. Moreover, SMARTUNIFIER Manager comes with a built-in Monitoring Dashboard that allows monitoring of the distributed SMARTUNIFIER Instances.

**Does SMARTUNIFIER support Monitoring integration?**

Yes, this is possible; Each SMARTUNIFIER Instance creates detailed logs that document every communication activity. These logs can be made accessible to any external system e.g., by a dedicated monitoring communication channel. In addition, SMARTUNIFIER is able to send any kind of monitoring message (e.g., based on status changes or other events (e.g., time triggered) to any (or multiple) upper level monitoring system in any required format.

**Does SMARTUNIFIER provide certificate handling?**

Yes, SMARTUNIFIER is able to handle certificates and establish state-of-the-art secured connections (e.g., TLS, secured MQTT, secured OPC-UA, etc.).

**Is it possible with SMARTUNIFIER to limit access to data?**

Yes, SMARTUNIFIER Instances work on independent Windows/Linux computer units. Data may be stored temporarily on these HW devices as logs or for buffer (cache) purposes. This temporary data can be protected by assigning the HW with appropriate access rights and user roles.

**Does SMARTUNIFIER support services for security supervision and security monitoring?**

Yes, SMARTUNIFIER creates detailed logs regarding all communication activities (and other activities) it performs. With SMARTUNIFIER it is possible to integrate with any external security supervision/monitoring system (e.g., Splunk) and provide on-line log files (in any required format) to these systems by usage of a dedicated monitoring communication channel.

**Does SMARTUNIFIER support End-to-End transport encryption (to Northbound and Southbound)?**

Yes, SMARTUNIFIER is capable of supporting End-to-End transport encryption for southbound and northbound communication channels.

**Does SMARTUNIFIER enforce secure individual authentication for all users?**

Yes, SMARTUNIFIER supports individual user authentication.

**Does SMARTUNIFIER enable Windows AD / LDAP Integration?**

Yes, SMARTUNIFIER is capable of integrate with Windows AD / LDAP on customer request. An easy-to-use configuration capability will be provided soon.

**Does SMARTUNIFIER support a (configurable) secure remote access?**

Yes, Secure remote access to SMARTUNIFIER Manager and SMARTUNIFIER Instances is possible by standard Windows or Linux tools (e.g., ssh).

**Can SMARTUNIFIER protect unsecured Shop Floor devices from office network through isolation?**

Yes, a SMARTUNIFIER Instance can be deployed locally near an equipment/device and map any unsecured equipment/device interface into a secured protocol (e.g., OPC-UA, MQTT). This way “unsecured data streams” coming from an equipment/device can be transferred to any northbound system in a secured way (isolation of the equipment/device). The same principle can be also applied when sending control parameters (e.g., screw parameters, NC programs, recipes, ...) or commands from a northbound system to the equipment/device.

**Does SMARTUNIFIER support malware protection concepts (e.g., support of standard Anti-Virus Software)?**

Yes, SMARTUNIFIER works with any standard malware protection software incl. McAfee, NOD and many others.

**Is SMARTUNIFIER secure by design (e.g., secure coding guidelines, use of open source code, pen-testing)?**

SMARTUNIFIER was developed according state-of-the-art coding principles and on request we are willing to let perform any checks, verifications, pen testing as required to validate the software. Especially for realizing communication channels and implementing protocols, state-of-the-art Open Source Libraries are used and constantly updated to the newest versions available.

**Does SMARTUNIFIER support a range of transmission/infrastructure protocols (e.g., IPV4/IPv6)?**

Yes, with SMARTUNIFIER (depending on used HW) IP4/IP6 are supported.

- LAN: Up to 4x Gbit Ethernet Intel i211
- Wireless LAN: 802.11ac dual antenna + BT 4.2
- Cellular communication: LTE/WCDMA/GSM/GNSS

USB: Up to 8 ports, 2x USB 3.0, Up to 6x USB 2.0

- RS232 serial port

Also other transmission/infrastructure protocols can be supported on request but may require additional HW.

**Does SMARTUNIFIER provide the ability to handle intermittent connectivity of sources (data/event redelivery and failure modes)?**

Yes, intermittent connectivity of sources can be handled by SMARTUNIFIER Communication Channels. Based on rules, data/event redelivery can take place, failure modes can be activated, and escalation procedures to northbound systems can be triggered.

**Does SMARTUNIFIER reduce unnecessary traffic on shop floor network to protect device interfaces from traffic overload?**

Yes, a SMARTUNIFIER instance can be deployed locally nearby the equipment on any suitable HW device. The SMARTUNIFIER instance can then be configured to communicate to the connected southbound equipment/devices by using a separate physical network port and this way isolate the device from unnecessary traffic coming from the northbound network.

**Does SMARTUNIFIER support low Latency between Southbound and Northbound Interfaces?**

Yes, SMARTUNIFIER provides high performance / low latency by its distributed architecture consisting out of small SMARTUNIFIER Instances (i.e. no central bottlenecks like e.g., a middleware broker/database). Furthermore, SMARTUNIFIER features an integrated compiler that creates native Bytecode for the interfaces to be executed within the SMARTUNIFIER Instances. This makes the SMARTUNIFIER highly performant, since no slow scripting language nor any slow interpreter is used to provide the connectivity functionality.

**Is it possible with SMARTUNIFIER to ensure a consistent setting of time stamps for events (NTP)?**

Yes, this is possible.

**Is it possible to use UNICODE for operational data?**

Yes, it is possible to use UNICODE with SMARTUNIFIER (e.g., for OPC-UA).

### Is stability of SMARTUNIFIER s API given? Is the API stable across releases?

Yes, SMARTUNIFIER is a standard product from Amorph Systems. Interface stability is given and stable across new product releases. Furthermore interfaces are versioned and under controlled release management (i.e. different versions of interface, Information, Models and Mappings can be maintained and deployed in a controlled mode).

### Which tools for development, deployment and error analysis can be used with SMARTUNIFIER ?

For extension, deployment and error analysis of SMARTUNIFIER (e.g., development of new Information Models, pre-processing, aggregation etc.) widely-used and accepted state-of-the-art development environments and powerful standard tools may be used, e.g., Eclipse, Maven/sbt, Jenkins, Docker. For Error Analyses detailed logs created by SMARTUNIFIER can be used and analyzed with any analytics tools.

### What is the cost model of SMARTUNIFIER ?

Please refer to Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com)) for prices for SMARTUNIFIER . In general, the following policies apply:

- SMARTUNIFIER Manager is free of charge
- For SMARTUNIFIER Instances a yearly license fee is charged

### Does Amorph Systems offer reliable support for SMARTUNIFIER ?

For many years, Amorph Systems is providing first class support and intensive care to all of its customers. This covers all products and solutions that were delivered and operated in Industrial Areas as well as in Air Traffic Industry. For customer references please refer to Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com)).

### What support levels (SLAs) are supported?

Different levels of services (8x5, 8x7 up to 24x7) are available upon request from Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com)).

### Does SMARTUNIFIER support multiple languages?

Yes, SMARTUNIFIER is capable of supporting multiple languages. Currently the GUI is available in English and German language. In case more languages are required, please contact Amorph Systems ([www.amorphsys.com](http://www.amorphsys.com))

**Does Amorph Systems provide relevant training capabilities for operating SMARTUNIFIER and for engineering of Information Models and Mappings?**

Yes, SMARTUNIFIER is a simple to use standard product and was specifically designed as a powerful tool to enable the end customers themselves to provide seamless equipment/device as well as IT-Systems interconnectivity within their industrial environments.

Therefore, Amorph Systems trains customers to configure, deploy and operate SMARTUNIFIER in their environments. Moreover we can give advanced trainings, so that the customers can also implement new interfaces, new channels, new, Information Models and new Mappings on their own.