
MORPH.pro **SMARTUNIFIER**

SMARTUNIFIER User Manual

Release 1.4.0

Amorph Systems GmbH

Sep 18, 2021

ABOUT SMARTUNIFIER

1	About SMARTUNIFIER	2
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
2	How to integrate with SMARTUNIFIER	16
2.1	Information Models	16
2.1.1	What are Information Models	16
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
2.2.4	General Configurations	49
2.3	Mappings	51
2.3.1	What are Mappings	51
2.3.2	How to create a new Mapping	51
2.3.3	How to create Rules	53
2.4	Device Types	62
2.4.1	What are Device Types	62
2.4.2	How to create a new Device Type	63
2.5	Communication Instances	65

2.5.1	What are Instances	65
2.5.2	How to create a new Instance	66
3	Configuration Component Management	68
3.1	Naming Convention	68
3.2	Group Filter	68
3.3	Component Version Control	69
3.3.1	How to release configuration components	70
3.4	Operations	70
3.4.1	Add	70
3.4.2	Edit	71
3.4.3	Apply	72
3.4.4	Save	72
3.4.5	Save and Close	72
3.4.6	Search	73
3.4.7	Sort	74
3.4.8	Reload	74
3.4.9	Import	75
3.4.10	Export	77
3.4.11	Clone	78
3.4.12	Delete	78
3.4.13	Exit Editing	79
4	Deployment	80
4.1	What is a Deployment	80
4.2	Deploy Locally	81
4.3	Deploy with Docker	82
4.4	Deploy with AWS Fargate	84
4.4.1	Prerequisites	84
4.4.2	Architecture	88
4.4.3	Planning the Deployment	90
4.4.4	Deployment Steps	91
4.5	How to deploy, run and operate a deployed Instance	94
4.5.1	How to deploy an Instance	94
4.5.2	How to run an Instance	94
4.5.3	How to stop an Instance	95
4.5.4	How to delete a Deployment of an Instance	95
4.5.5	How to un-deploy an Instance	95
4.6	How to monitor a deployed Instance	96
4.6.1	Log Viewer	96
4.6.2	Dashboard	97
5	Administration	99
5.1	Active Directory Integration (ADI)	99
5.1.1	AD Group Mapping	100
5.2	Backup and Restore	102
5.2.1	Backup	103
5.2.2	Restore	104

5.2.3	Manager Backup	105
5.3	Channel Types Manager	106
5.3.1	About Layers	106
5.3.2	How to create a new Channel Type	107
5.4	Docker Java Image Manager	109
5.4.1	Access to the Docker Java Image Manager	109
5.4.2	Add a New Docker Java Image	111
5.4.3	Edit a Docker Java Image	111
5.4.4	Delete a Docker Java Image	112
5.5	Deployment Endpoints	112
5.5.1	What are Deployment Endpoints	112
5.5.2	Deployment Endpoints Types	112
5.6	User Management	116
5.6.1	About User Management	116
5.6.2	Add a new user	116
5.6.3	Edit a user	119
5.6.4	Delete a user	120
6	Demo Scenarios	123
6.1	File-based data - CSV to REST-Server	123
6.1.1	Overview	123
6.1.2	Information Model	125
6.1.3	Communication Channel	126
6.1.4	Mapping	127
6.1.5	Device Type	128
6.1.6	Instance	129
6.1.7	Deployment	131
6.2	File-based data - Insert JSON data in SQL-Database	132
6.2.1	Overview	132
6.2.2	Prerequisite	132
6.2.3	Information Model	133
6.2.4	Communication Channel	134
6.2.5	Mapping	136
6.2.6	Device Type	137
6.2.7	Instance	137
6.2.8	Deployment	138
6.3	File-based data - XML, Database, and MQTT	138
6.3.1	Overview	138
6.3.2	Prerequisites	139
6.3.3	Information Model	140
6.3.4	Communication Channel	141
6.3.5	Mappings	145
6.3.6	Device Type	146
6.3.7	Instance	146
6.3.8	Deployment	147
7	Getting Help	148
7.1	Troubleshooting	148

7.1.1	Communication Instance works abnormally or crashes	148
7.1.2	Manager works abnormally or crashes	149
7.2	FAQ	149
7.3	Glossary	158

**Integrate perfectly your
Production-IT using**

SMARTUNIFIER
ADVANCED IT-INTEGRATION PLATFORM

AMORPH.pro
SMARTUNIFIER

ABOUT SMARTUNIFIER

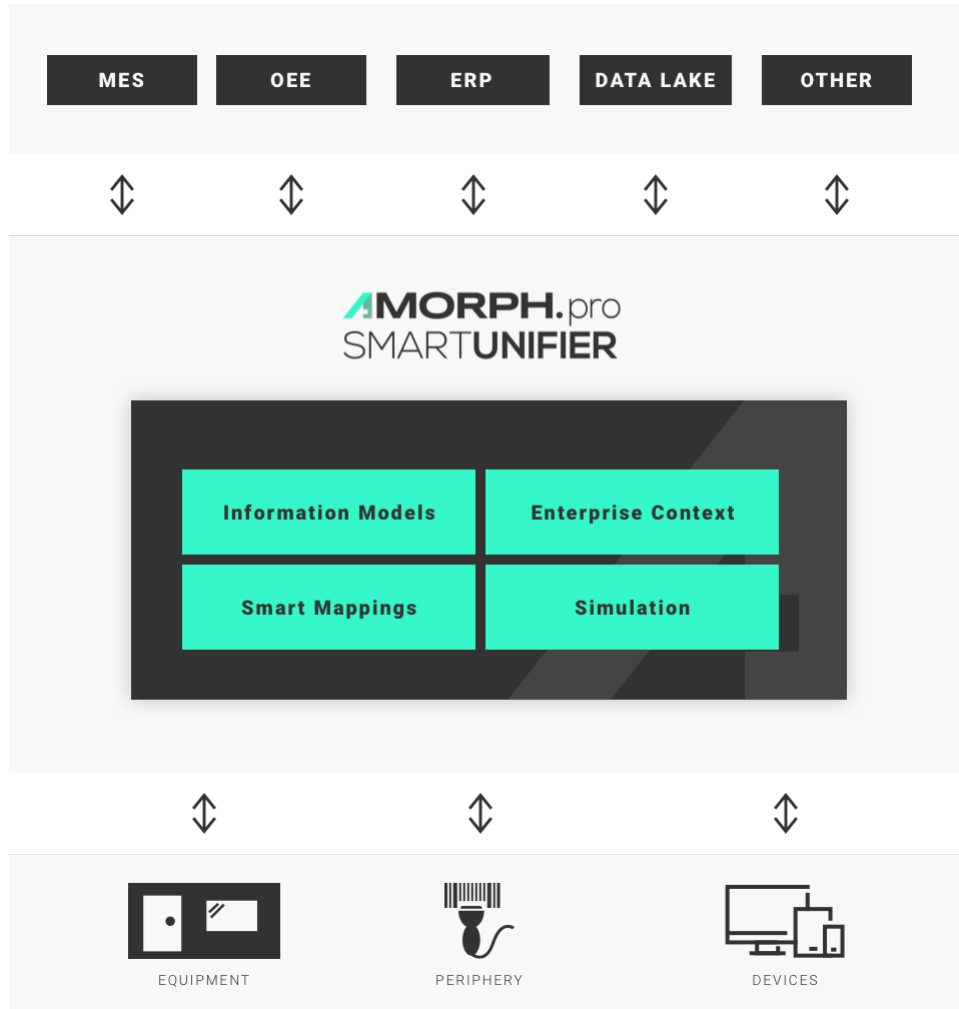
You are new to SMARTUNIFIER?

- 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*.

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 can 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

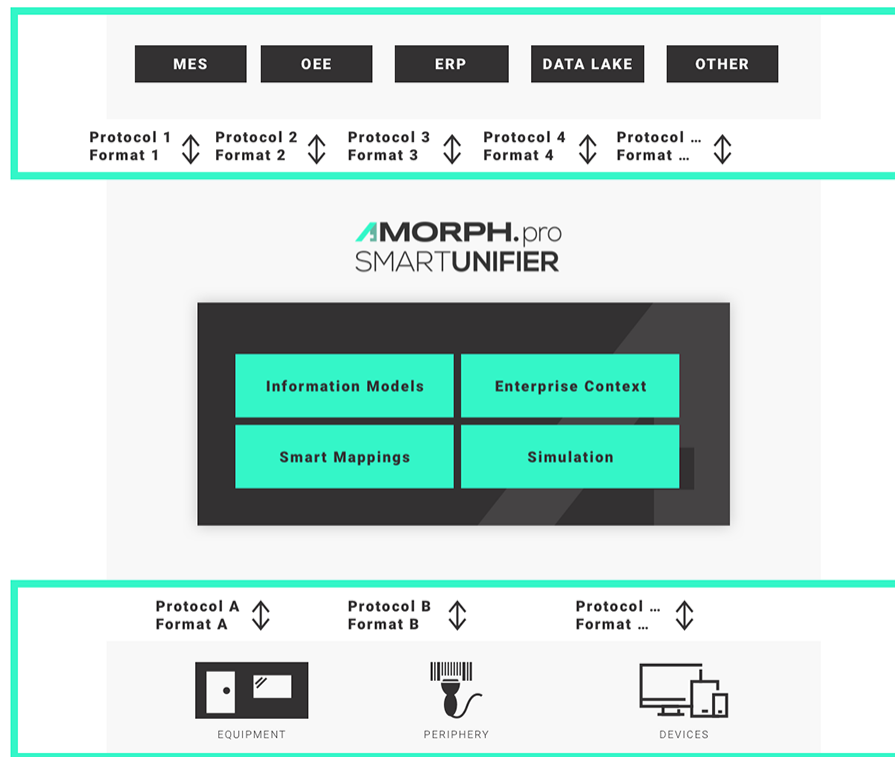
Easy, fast and flexible bi-directional interconnection of multiple IT systems and equipment within a production facility.

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.



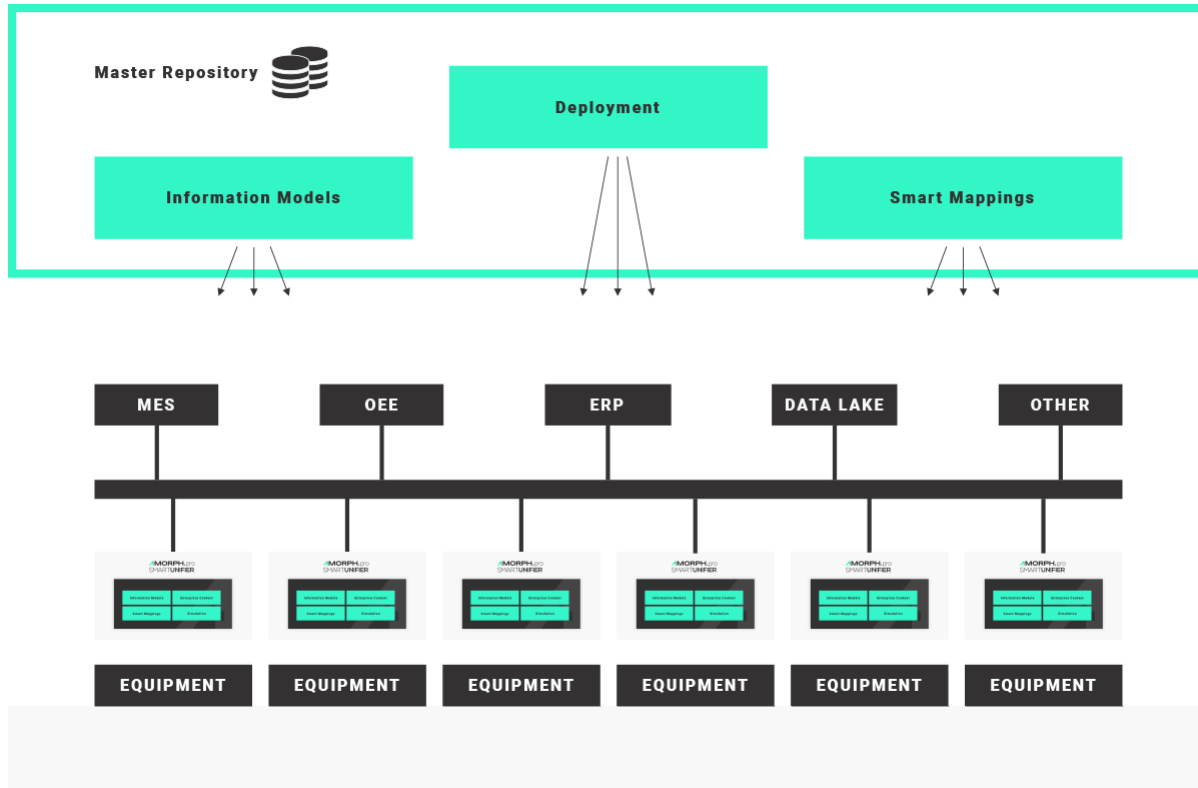
1.3.2 Reusable Interfaces and Interface Models

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.



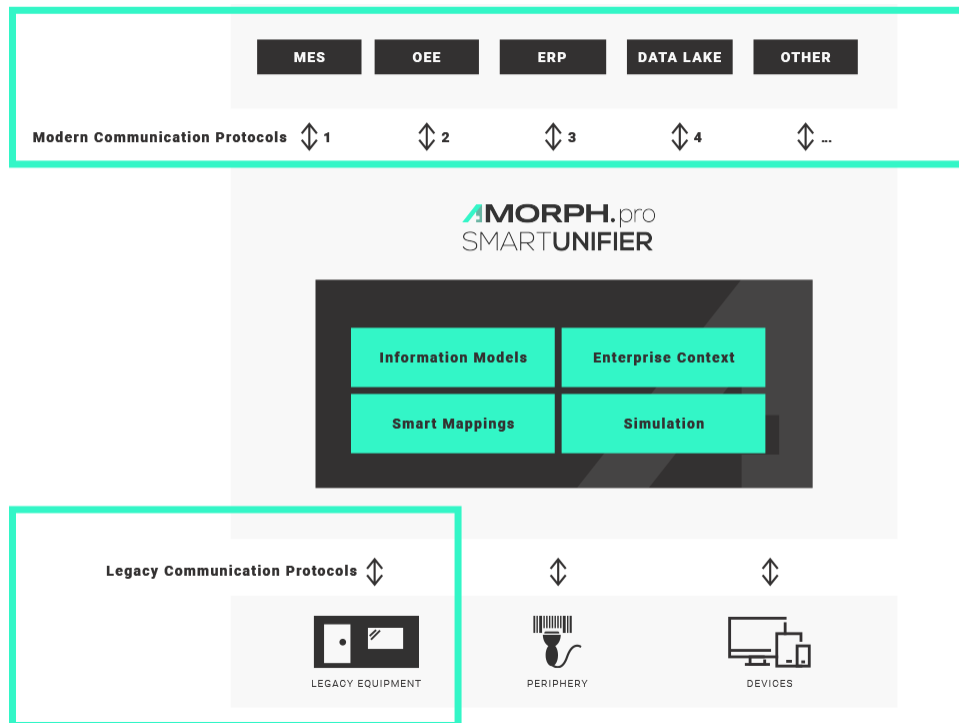
1.3.3 Integrate Legacy Equipment

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 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.

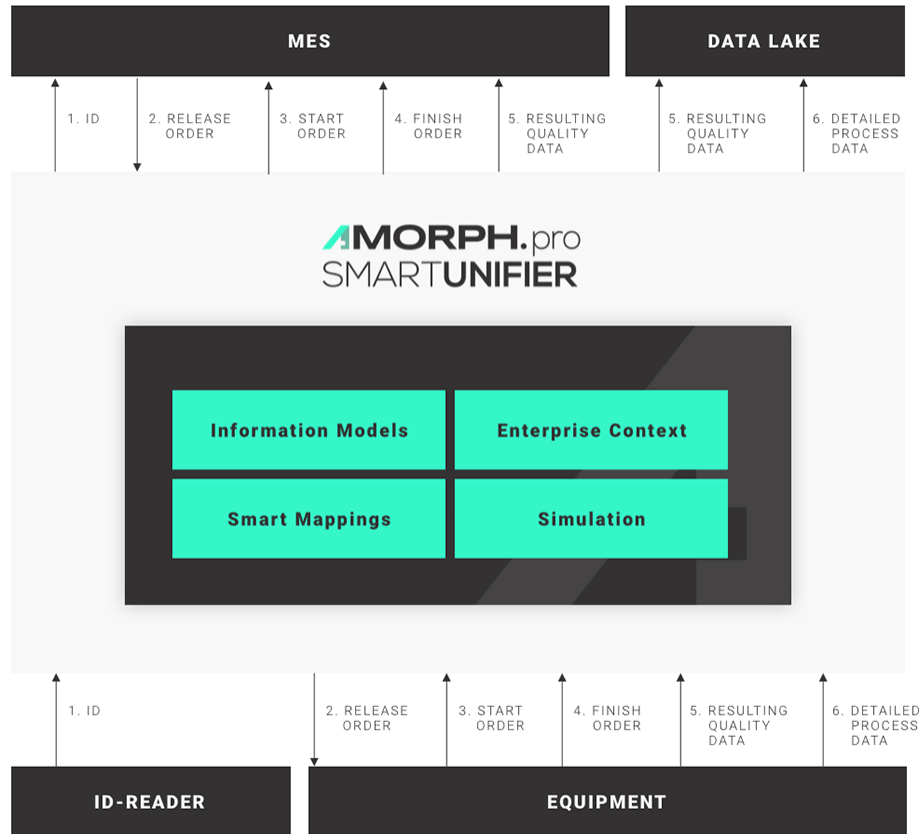


1.3.4 Implement Fab Communication Scenario

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

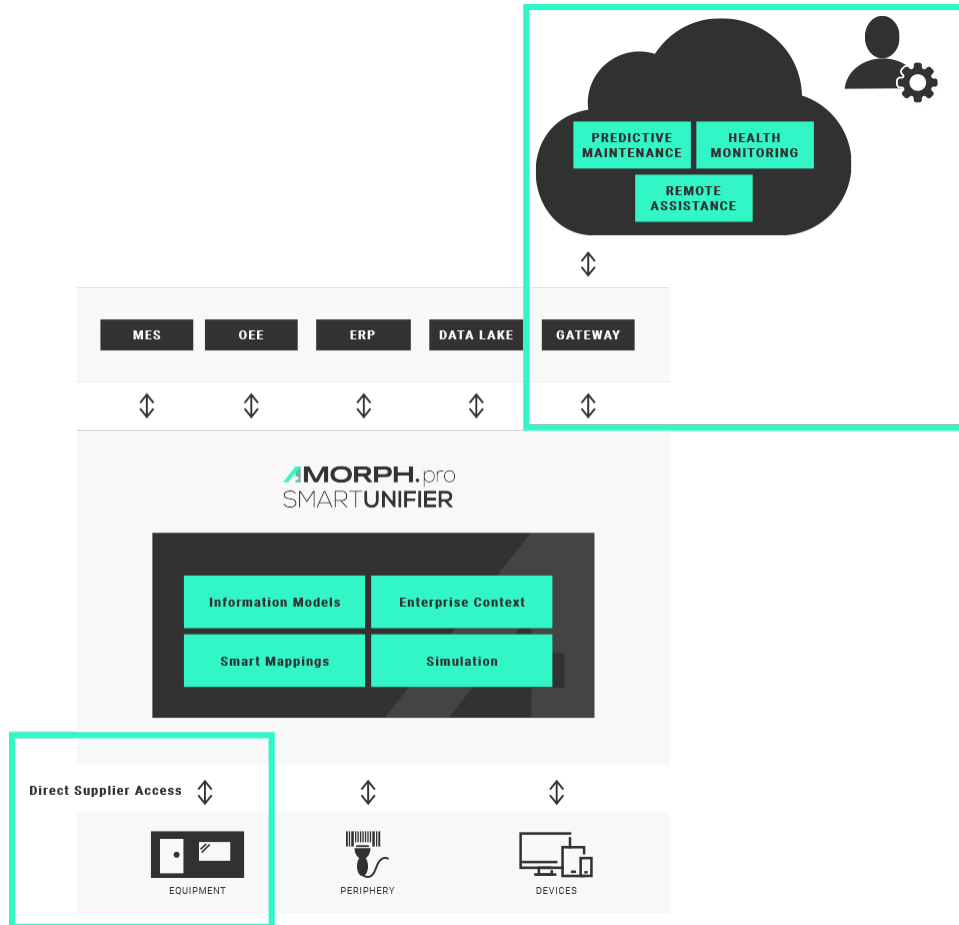
Establish new services and business models by giving secured multi-channel access to equipment and device data in real-time.

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 to achieve production optimization and to ensure the most efficient usage of equipment resources for customers.



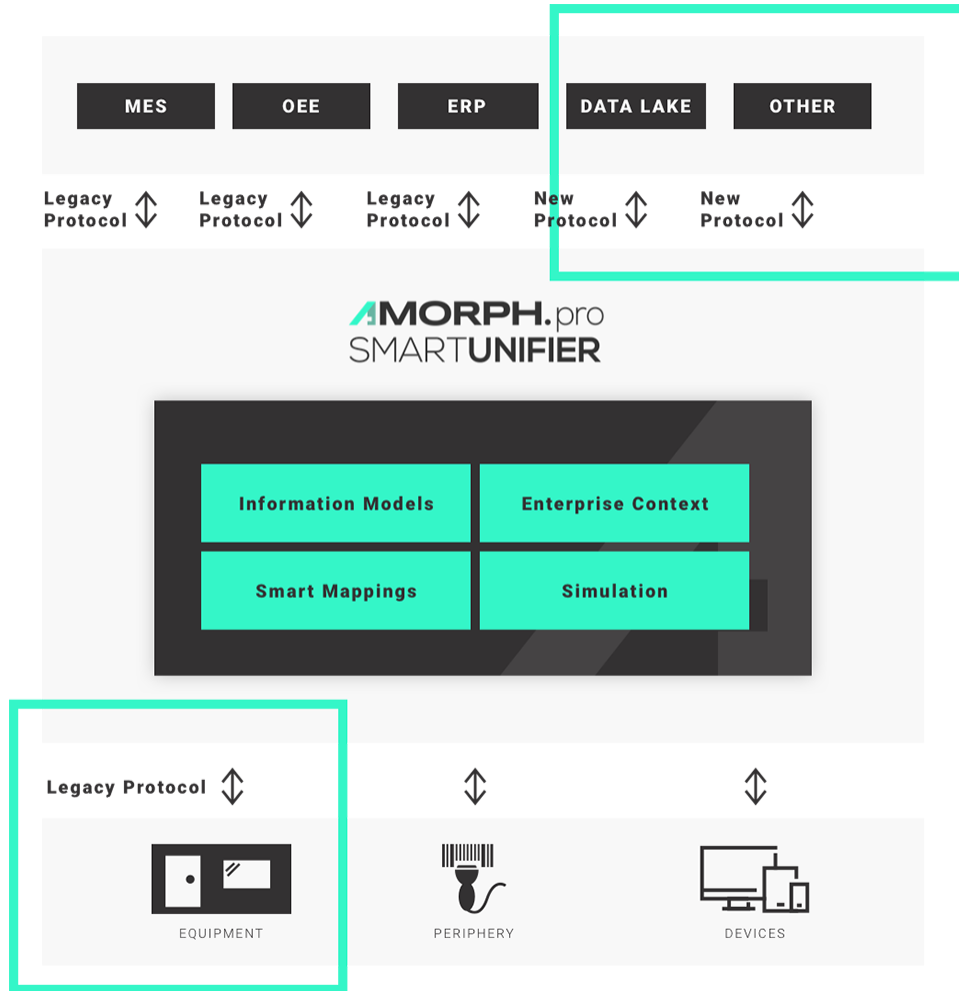
1.3.6 Migrate to Industry 4.0

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 incrementally 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.

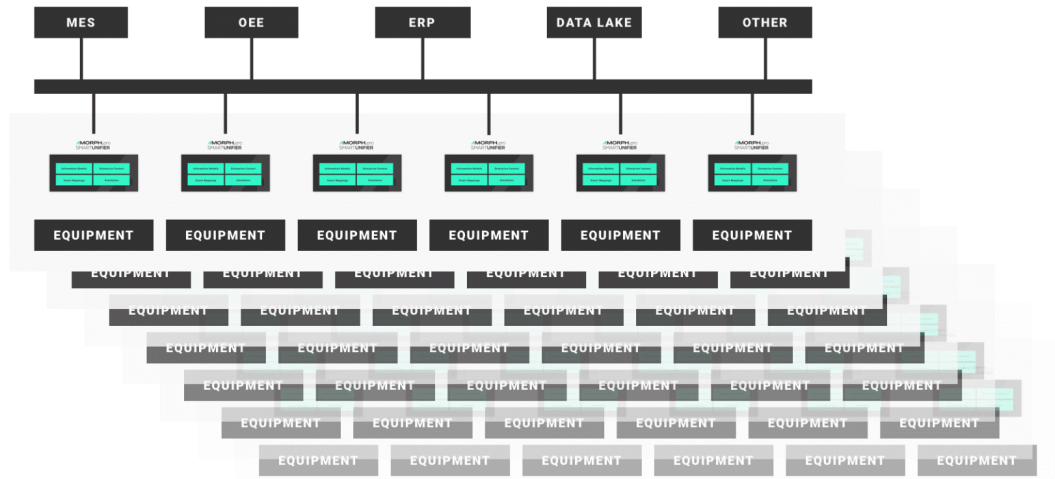
SMARTUNIFIER works as a distributed environment. Using advanced technologies of distributed computing is the key for enormous scalability.

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.

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.

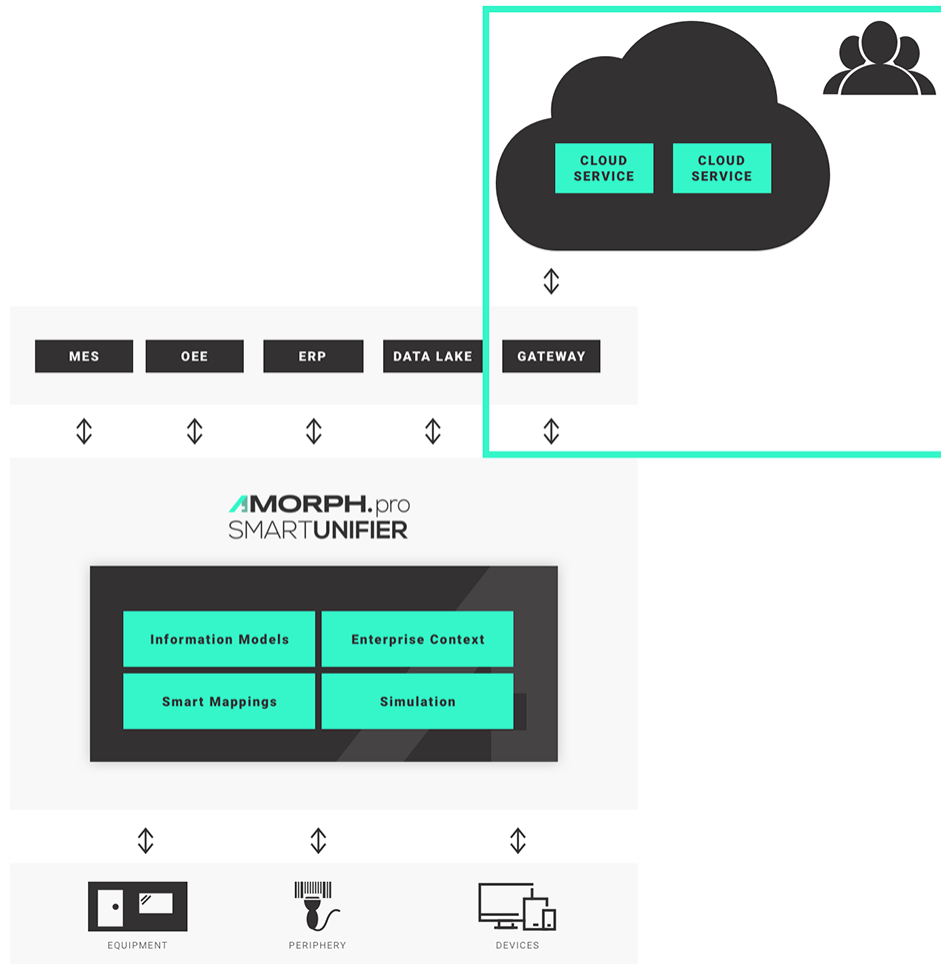
Thus, SMARTUNIFIER is an essential piece of Industry 4.0 for any manufacturing enterprise – allowing fab-wide and enterprise-wide management of production communication and IT integration infrastructure.



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)
EUROMAP	Connectivity of injection moulding machines via files
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
ISO-on-TCP	(RFC1006) Connectivity of S7 automation devices with any communication partner
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

continues on next page

Table 1 – continued from previous page

Format	Description
Modbus-TCP	Communication via Modbus TCP Server / TCP Client
Microsoft Azure (IoT Hub)	Interface to Microsoft Azure Iot Hub via MQTT
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

continues on next page

Table 1 – continued from previous page

Format	Description
SFTP	Upload and Download files to/from SFTP servers
UDP	Communication from/to any (binary) UDP based protocol
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

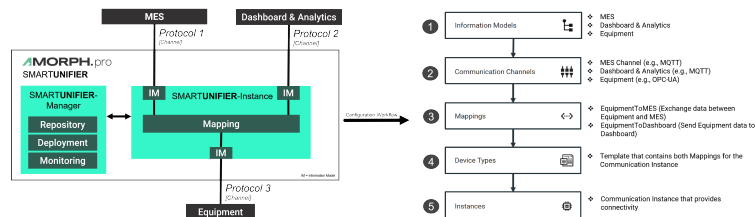
Note: In case a customer requires another data format not listed in the table, please contact Amorph Systems.

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.

Below you can see an example of integration scenario and the necessary steps to establish connectivity with SMARTUNIFIER:



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



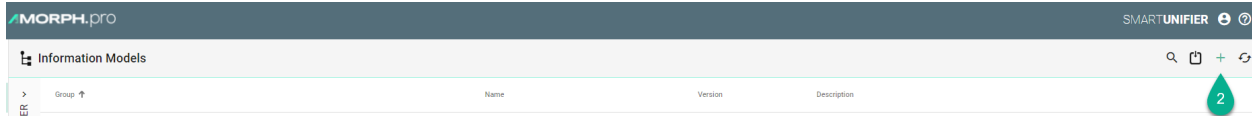
Deployment Endpoints



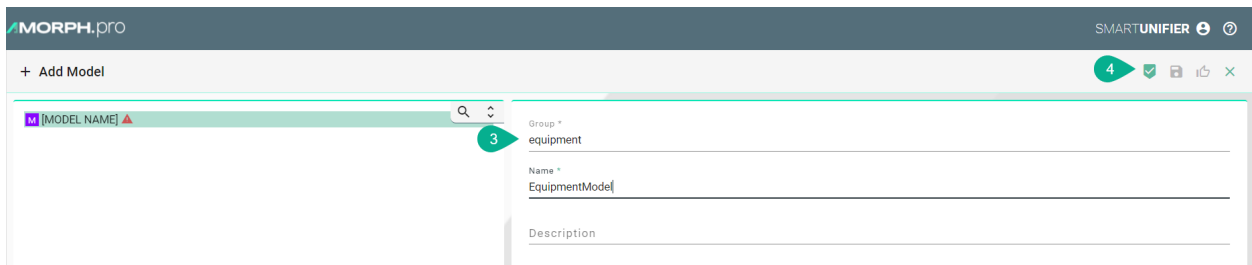
User Management



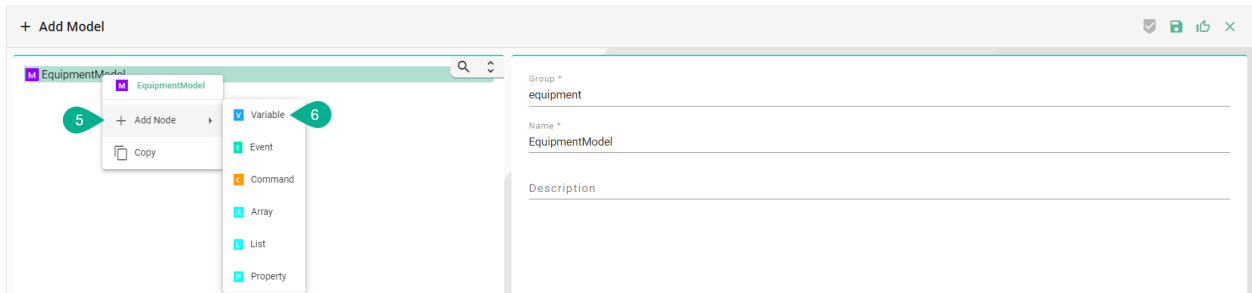
- You are presented with the following screen containing a list view of existing Information Models.
- In order to add a new Information Model, select the “Add Model” button at the top right corner (2).



- On the following screen provide the following mandatory information: Group and Name (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.

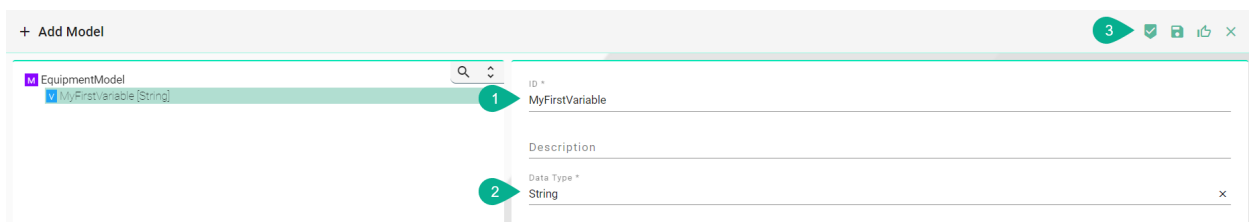
Variables

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 Data Type (2)
- Click the “Apply” button (3)



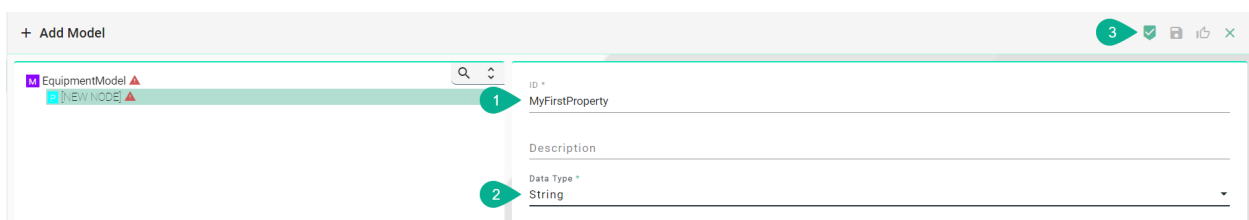
Properties

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 Property

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



Events

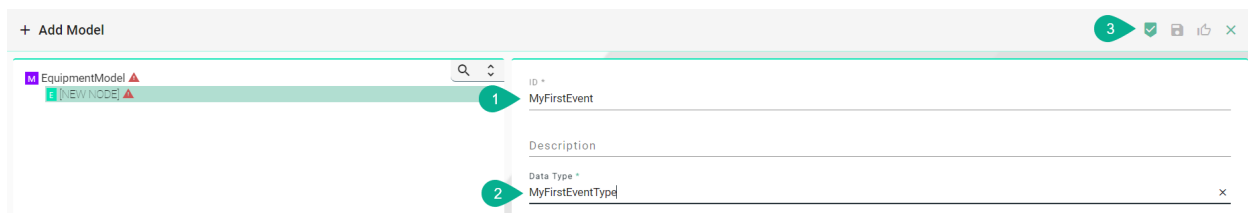
What are Events

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.

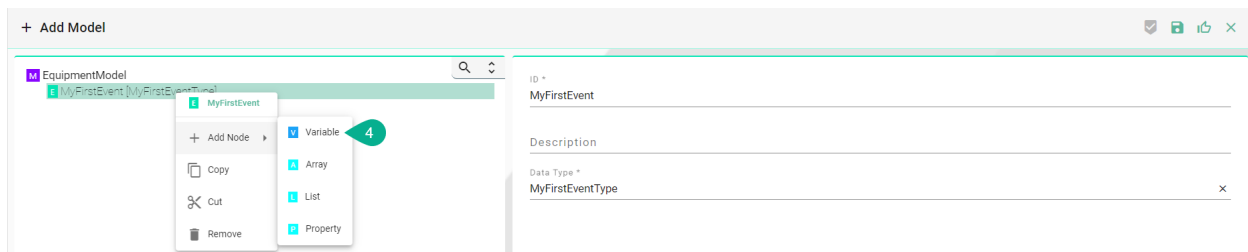
How to create an Event

- Enter an ID (1)
- Enter a Data Type for the Event. e.g., “MyFirstEventType” (2)
- Click the “Apply” button (3)

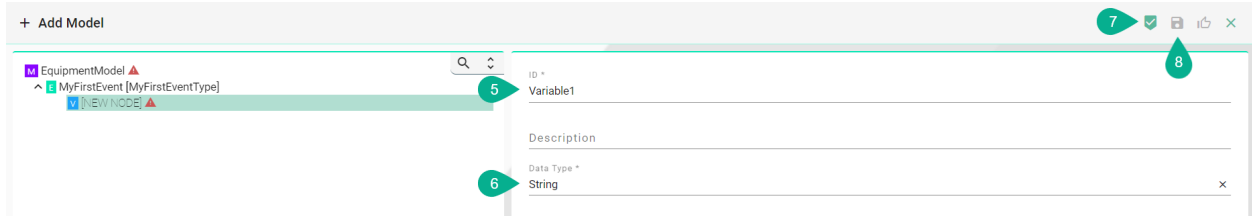


Within the Event *Variables*, *Arrays* or *Lists* can be added. Follow the steps below to add a Variable:

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



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



Commands

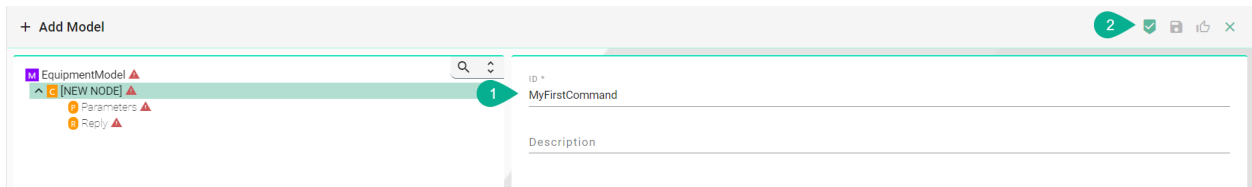
What are Commands

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

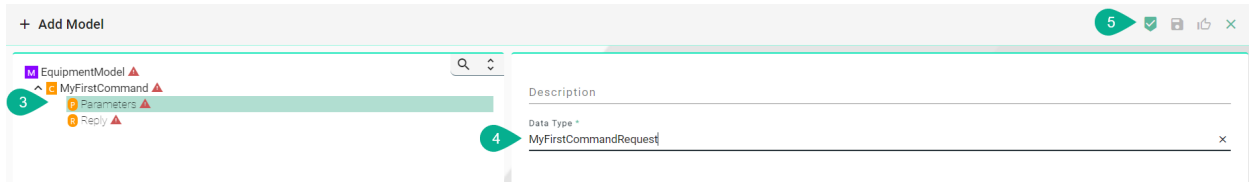
- Enter an ID (1)
- Click the “Apply” button (2)



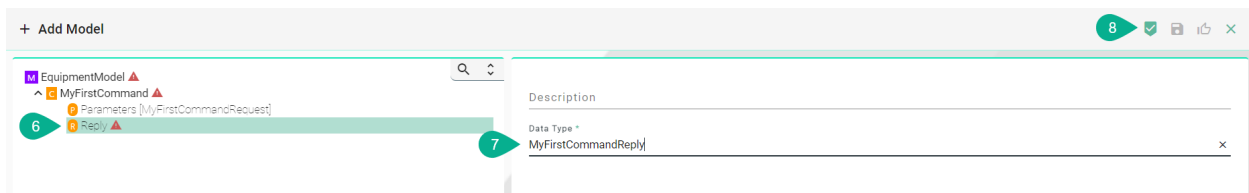
The main two parts of a Command are the Request, referred to as Parameters within the SMARTUNIFIER, and the Reply. *Variables*, *Arrays* and *Lists* can be added to both of these command parts.

Follow the steps below to add a Variable to Parameters:

- Select the Parameters node from the tree (3)
- Enter a Data Type (4)
- Click the “Apply” button (5)

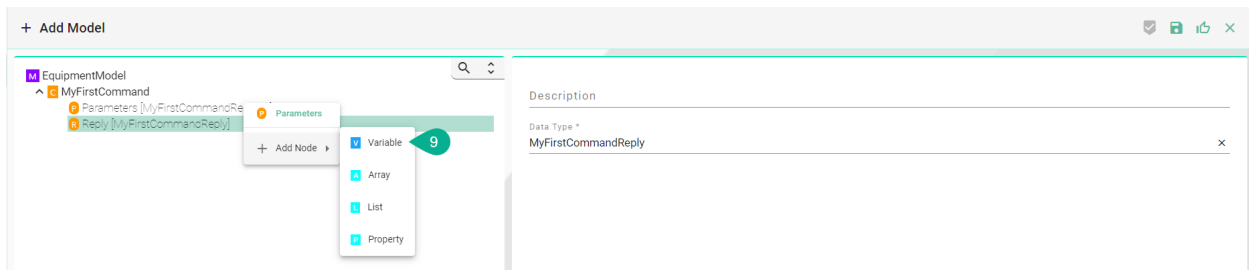


- Select the Reply node from the tree (6)
- Enter a Data Type (7)
- Click the “Apply” button (8)

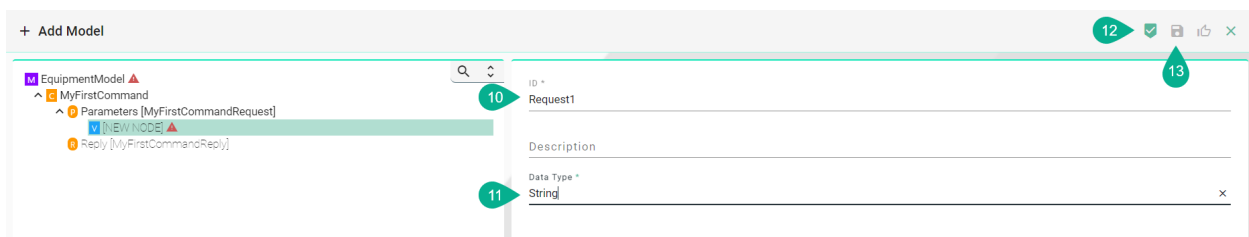


Follow the steps below to add nodes under the Parameter and Reply node:

- Right click the Parameter node, select “Add Node” and choose a Definition Type (9)



- Enter an ID (10)
- Enter a Data Type (11)
- Click the “Apply” button (12)
- Click the “Save” button (13) to save the Information Model



Arrays

What are Arrays

Arrays allow to hold a fixed size collection of elements, which have all the same data type. The size of the array must be defined in the configuration of the Information Model.

How to create an Array

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

The screenshot shows the 'Add Model' dialog in the SMARTUNIFIER application. On the left, a sidebar contains 'EquipmentModel' and 'NEW NODE' options. The main area is a form for creating a new model. It has a search bar at the top. Below it, there are four numbered green circles indicating the steps: 1. Enter an ID (MyFirstArray), 2. Select a Data Type (String), 3. Enter the size (5), and 4. Click the 'Apply' button (indicated by a green arrow pointing to the button in the top right corner).

Lists

What are Lists

Lists allow to hold a collection of elements (*Variables*), which can each have different data types.

How to create a List

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

The screenshot shows the 'Add Model' dialog in the SMARTUNIFIER application. On the left, a sidebar contains 'EquipmentModel' and 'NEW NODE' options. The main area is a form for creating a new model. It has a search bar at the top. Below it, there are two numbered green circles indicating the steps: 1. Enter an ID (MyFirstList), and 2. Enter a Data Type (String). A green arrow labeled '3' points to the 'Apply' button in the top right corner.

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)

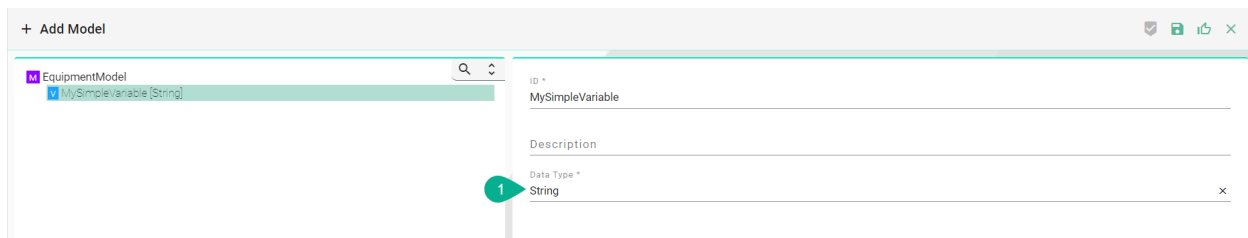


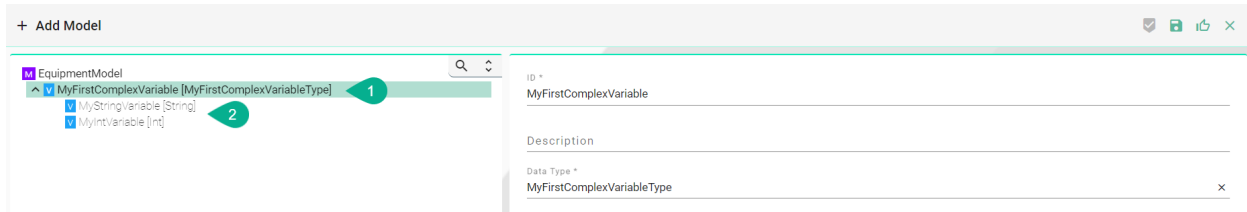
Table 1: Predefined Data Types

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.
LocalDate-Time	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.
OffsetDate-Time	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

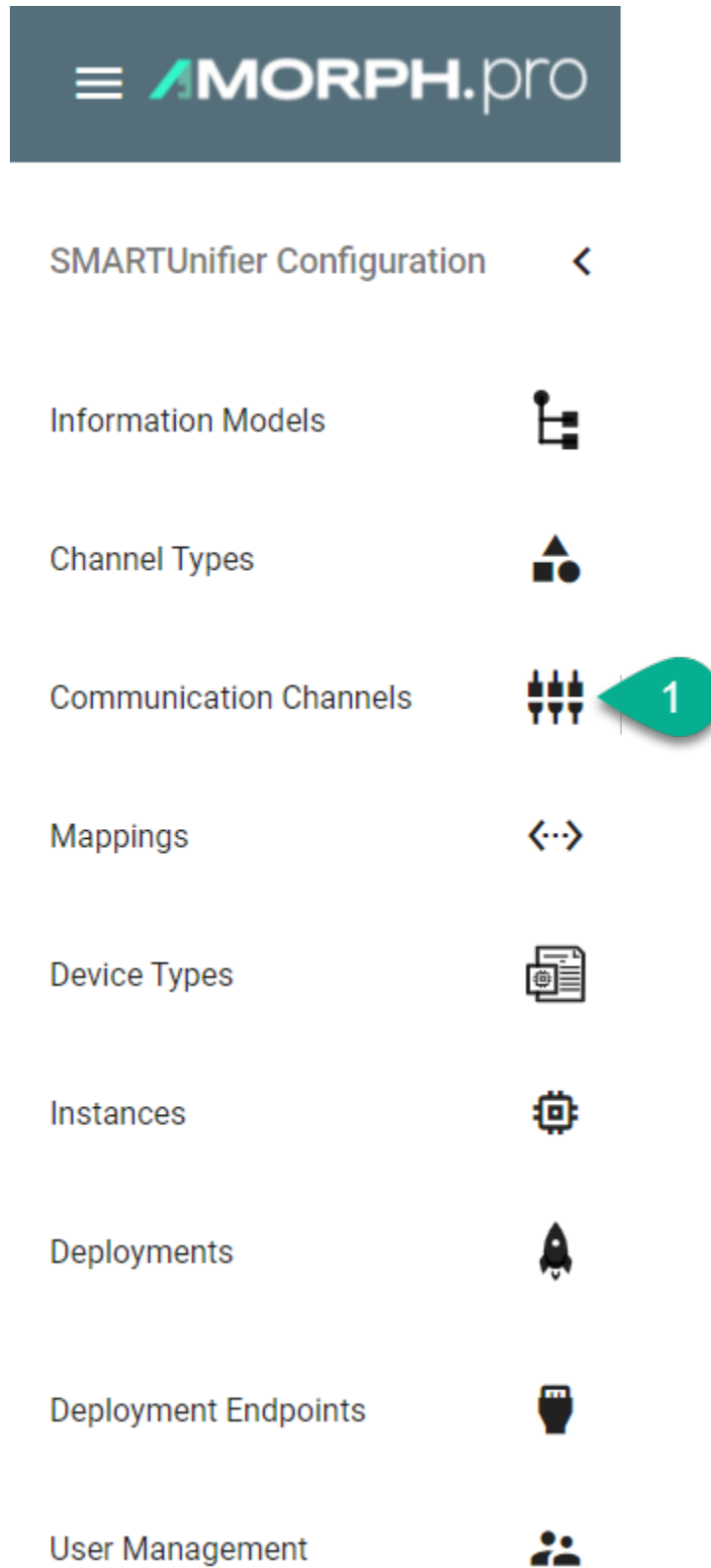
2.2.1 What are Channels

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.

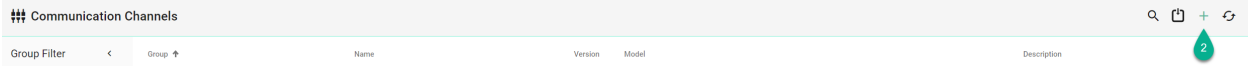
2.2.2 How to create a new Channel

Follow the steps below to create a new Channel:

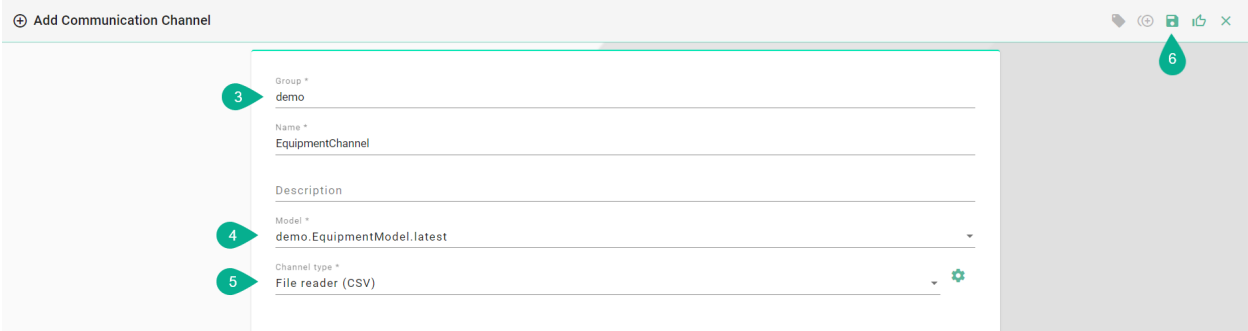
- Go to the Communication Channels perspective by clicking the “Communication Channels” button (1)



- 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



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

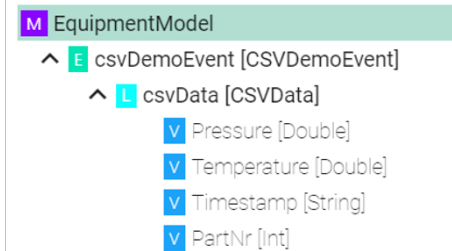
Information Model Requirements

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.

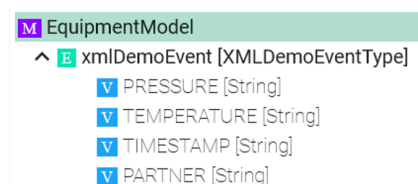
```
1 Pressure, Temperature, Timestamp, PartNr
2 17.5, 20, 2020.06.11-06:56:31, 0001
3 18.9, 22, 2020.06.11-07:56:31, 0002
```



XML

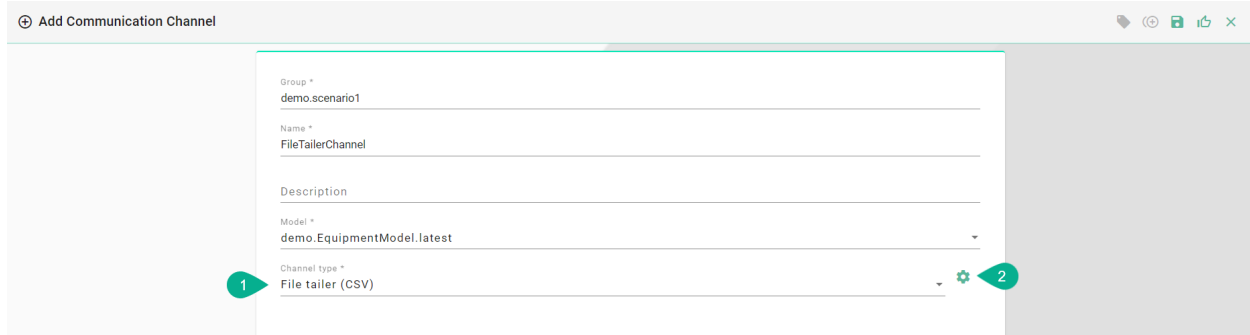
- Elements of the XML file are represented by the Node Type Variable **V**.
- Attributes of the XML file are represented by the Node Type *Property* **P**. In order to assign attributes to elements in the Information Model, the element Node Type **V** must be a *Custom Data Type*.

```
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>
```

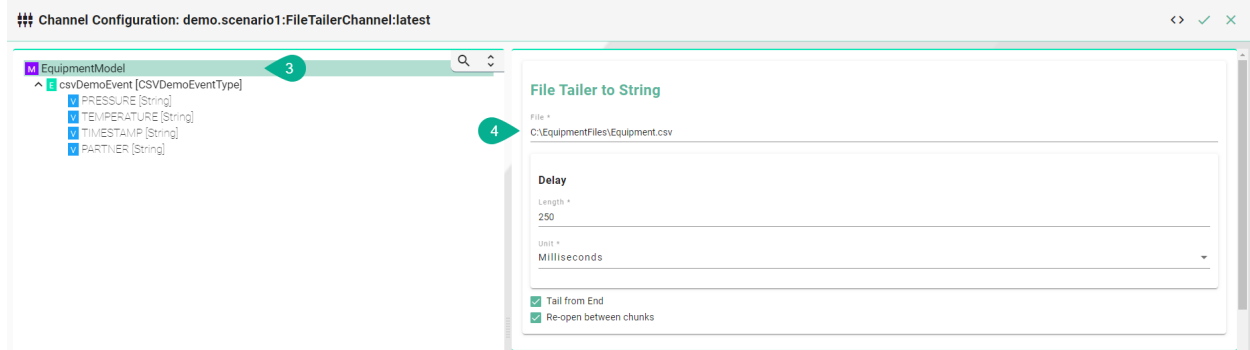


How to configure the File Tailer (CSV) Channel

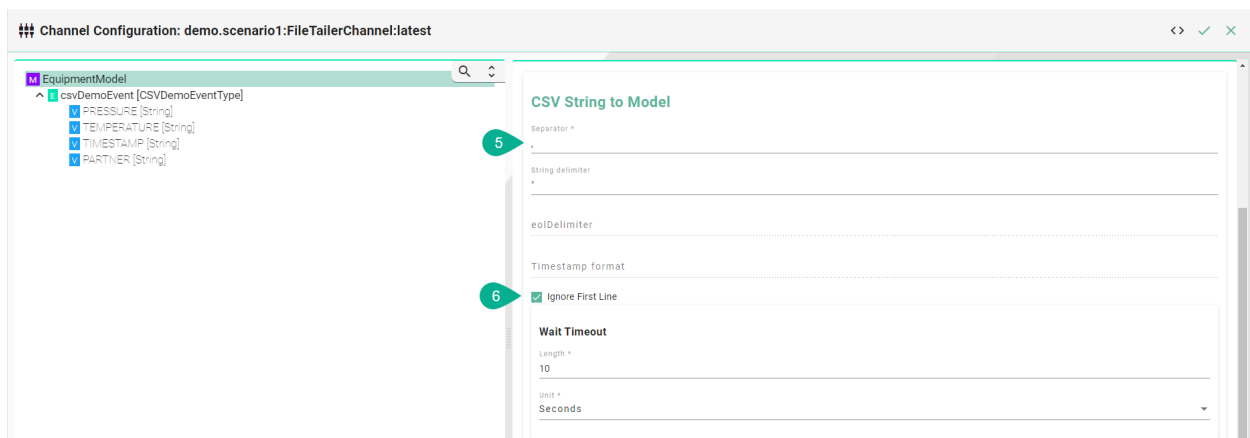
1. Select **File tailer (CSV)** from the Drop-Down.
2. Click the **Configure** button.



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.



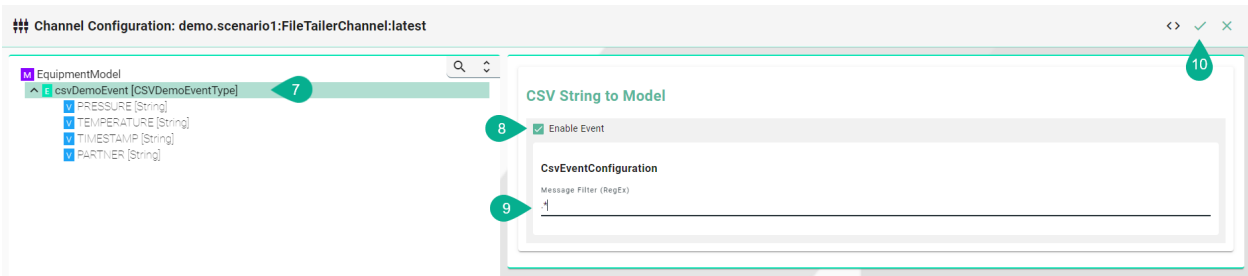
5. Enter the **separator** which is used in the CSV-file as well as the **string delimiter**, the **eol delimiter** and the **timestamp format** if one is used.
6. If the CSV file contains a header enable **ignoreFirstLine**.



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”	"
Eol Delimiter	Defining Carriage return and/or Line Feed	\r, \n
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 Reader

Characteristics

- File Reader 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:

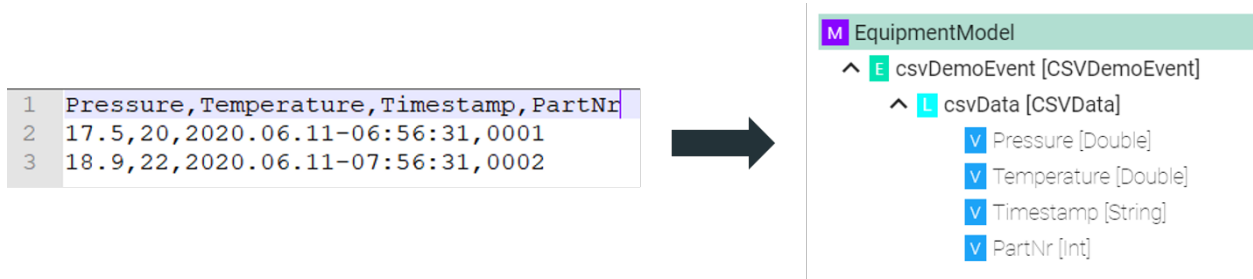
- CSV
- JSON
- XML

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.



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 **E** must be a *Custom Data Type*.

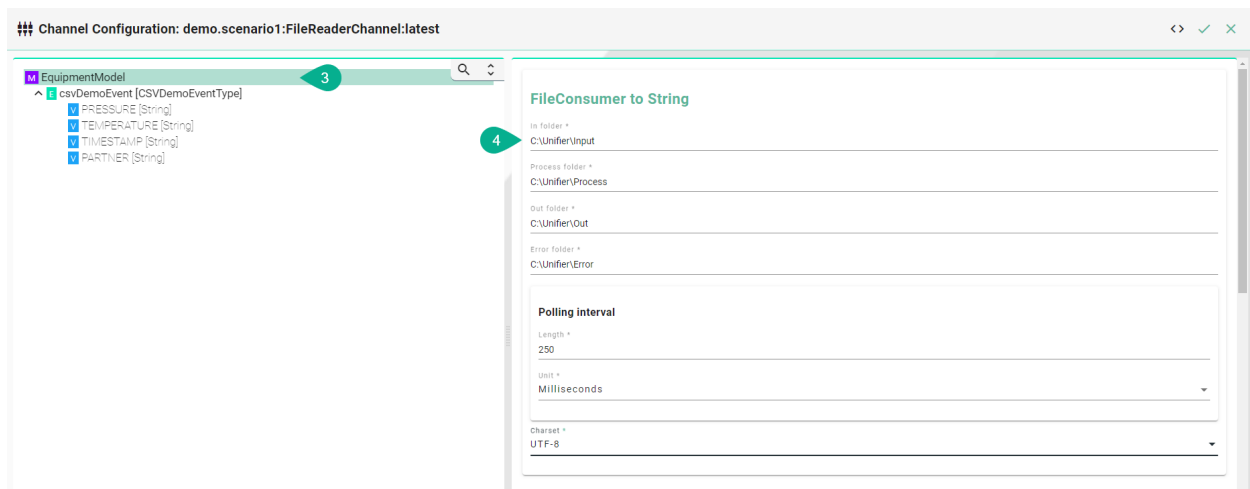


How to use File Reader with CSV

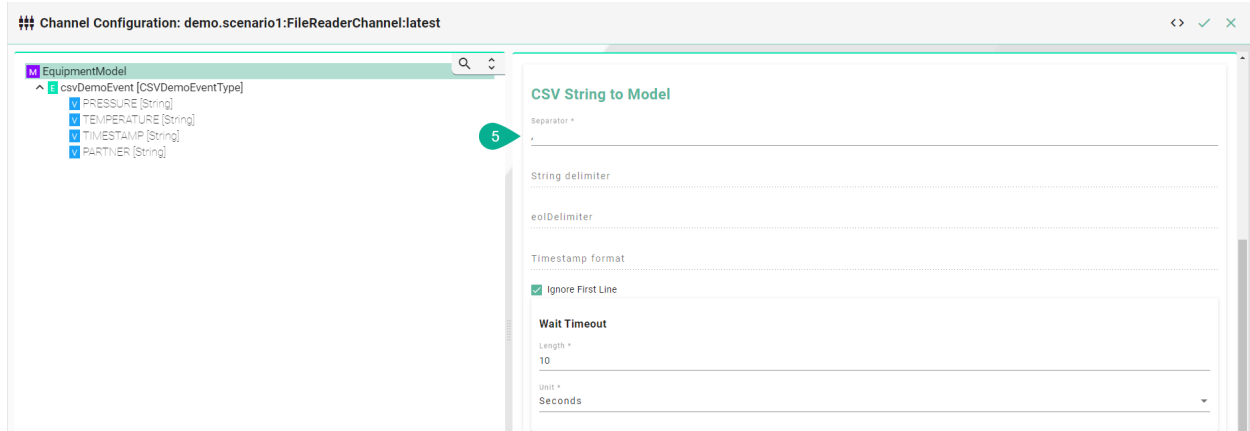
1. Select **File reader (CSV)** from the Drop-Down.
2. Click the **Configure** button.



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**
 - Specify the **polling interval**
 - Select the **CharSet** according to the file in use



5. CSV Consumer to Model - Configuration
 - Enter the **separator** which is used in the CSV-file
 - If needed: Set **string delimiter**, **eol delimiter** and the **timestamp format**
 - If the CSV file contains a header enable **ignoreFirstLine**



1. 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.

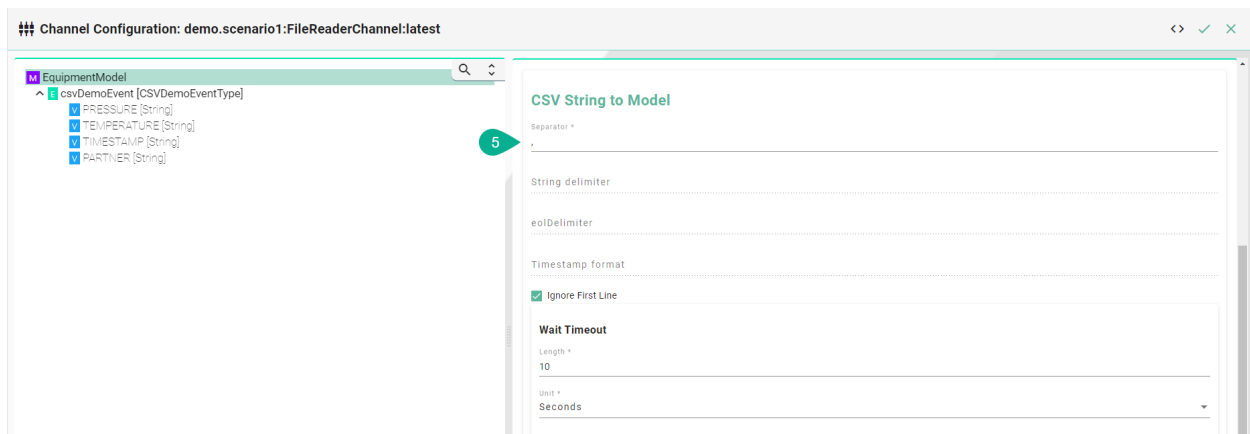
7. File Consumer to String - Configuration

- 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 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:

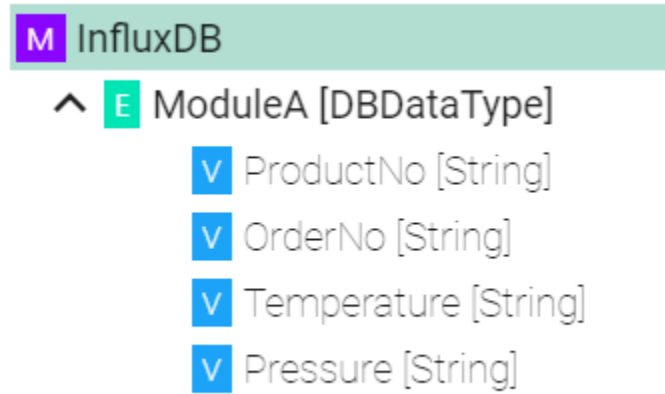
Property	Description	Example
Separator	Separator type, used in the csv file	, , ;
Delimiter	Values that have an additional delimiter like “Date”, “Time”	”
Eol Delimiter	Defining Carriage return and/or Line Feed	\r, \n
Timestamp format	Format of the timestamp	YYYY-MM-DD HH:mm:ss
ignoreFirst-Line	Delay 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 - InfluxDB**

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](#).

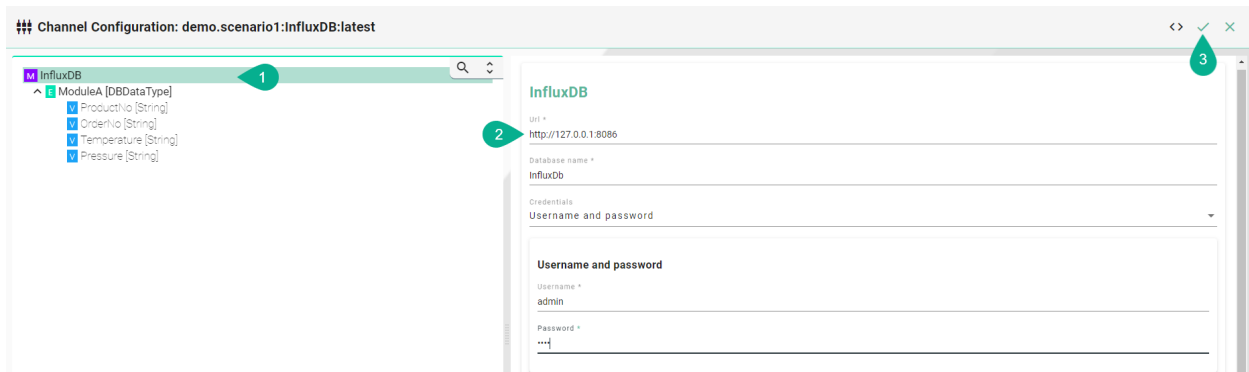
Information Model Requirements**Inserts**

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

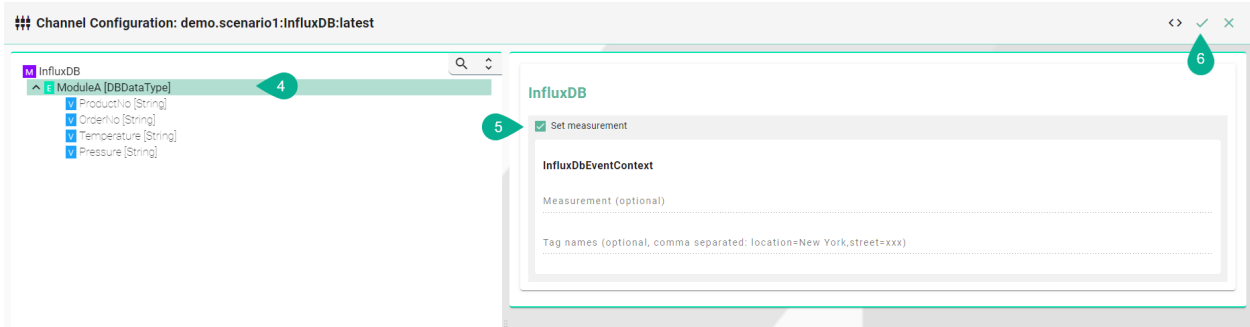


How to configure InfluxDB

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 **username** and the **password**
3. Click the **Apply** button and save the Channel by clicking the **Save** button on the upper right corner.



4. Select the **event node** in the tree on the left.
5. Configure Measurements (Tables within InfluxDB)
 - Enter a **measurement**
 - Enter **tag names**.
6. Click the **Apply** button and save the Channel by clicking the **Save** button on the upper right corner.



SQL Database

Characteristics - SQL Database

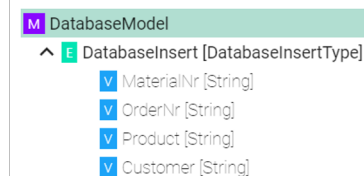
- The SQL Channel can be configured for the following two scenarios:
 - Inserting data
 - Updating data
 - Retrieving data
- When inserting values into the database please **note** that “infinity” values are converted automatically into “null” values.

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**.

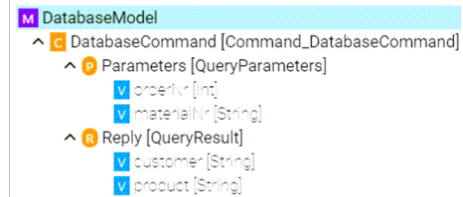
*	MATERIAL_Nr	ORDER_Nr	PRODUCT	CUSTOMER
1	HS787FSTC	121	AXY_200	DemoCompany1
2	HS787FSTC	123	AXY_150	DemoCompany2
3	HS777FSTC	120	AXY_100	DemoCompany1
4	HS767FSTC	123	AXY_200	DemoCompany1



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**.

*	MATERIAL_NR	ORDER_NR	PRODUCT	CUSTOMER
1	HS787FSTC	121	AXY_200	DemoCompany1
2	HS787FSTC	123	AXY_150	DemoCompany2
3	HS777FSTC	120	AXY_100	DemoCompany1
4	HS767FSTC	123	AXY_200	DemoCompany1



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 **username** and **password**.

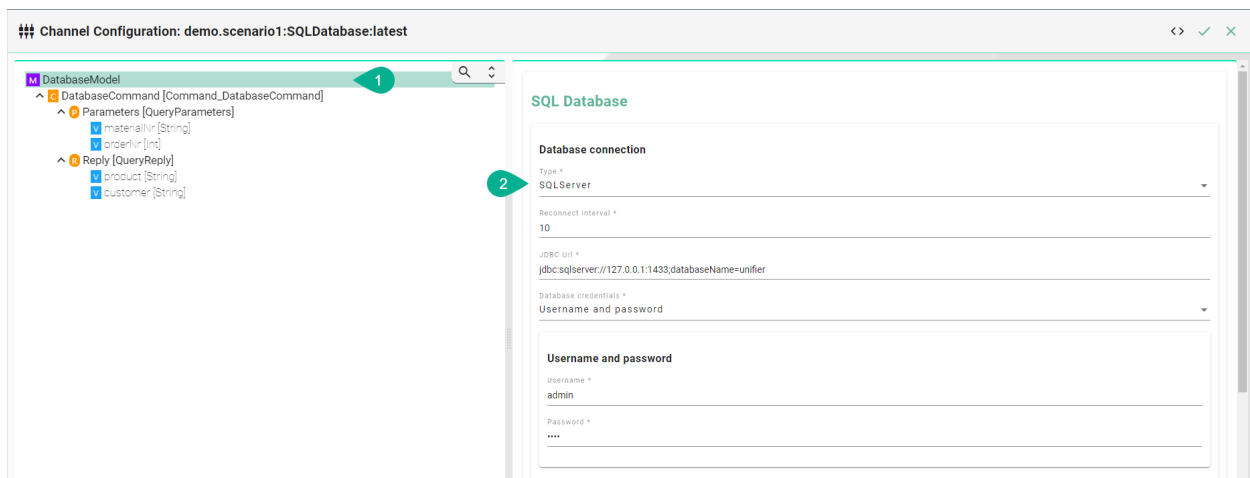


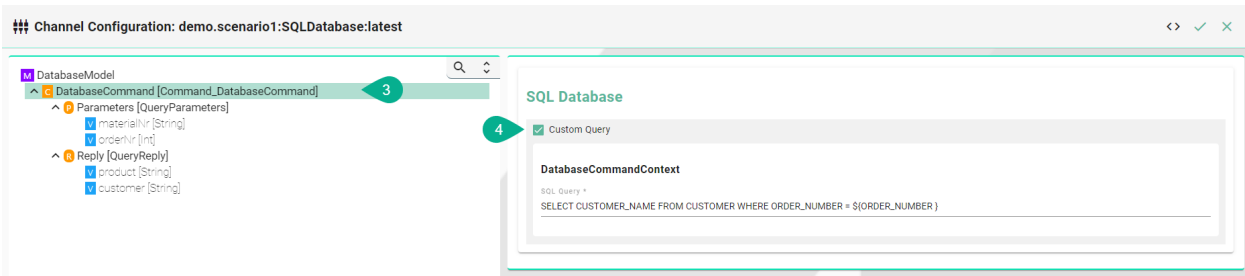
Table 2: Description of database configuration properties

Property	Description	Mandatory	Example
Type	Type of the database	Yes	MariaDB, SQLServer, ORACLE, HSQLDB, DB2, PostgreSQL
ReconnectInterval	Time to reconnect if connection fails	Yes	10 (in milliseconds)
JdbcUrl	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
Username and password	Credentials of the database	Yes	

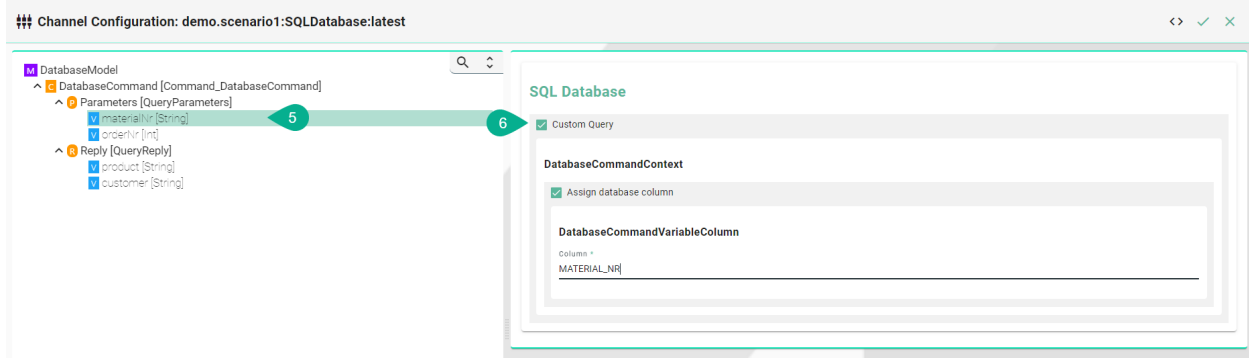
Note: The configuration of specific *information model nodes* differs whether you want to perform an **insert** or an **select** statement on the database. Inserting data into the database requires an **event node** whereas selecting data requires a **command node** in the *Information Model*.

Select Statement

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

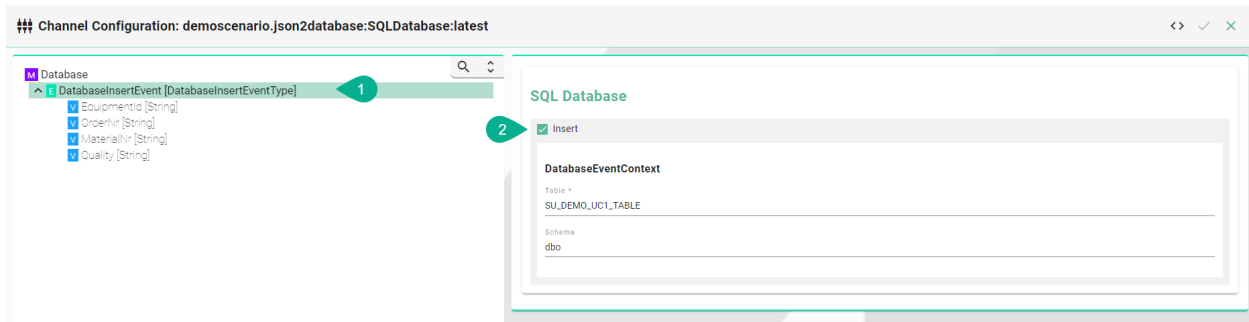


1. Each variable under *Parameters* and *Reply* needs to be assigned to a database column. Select the **variable node** under *Parameters* and in the tree select what needs to be configured.
2. Check the **assign database column** checkbox and enter the **column name** as it is defined in the used database.



Insert Statement

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



Protocols

MQTT

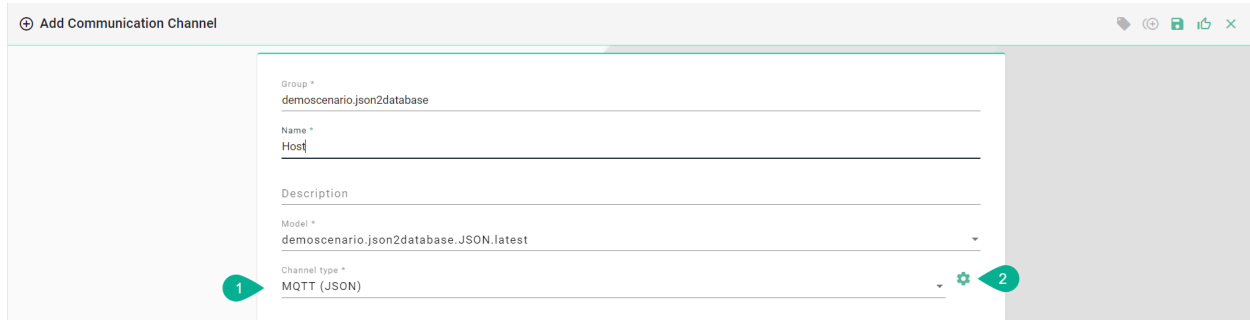
Characteristics - MQTT

Information Model Requirements

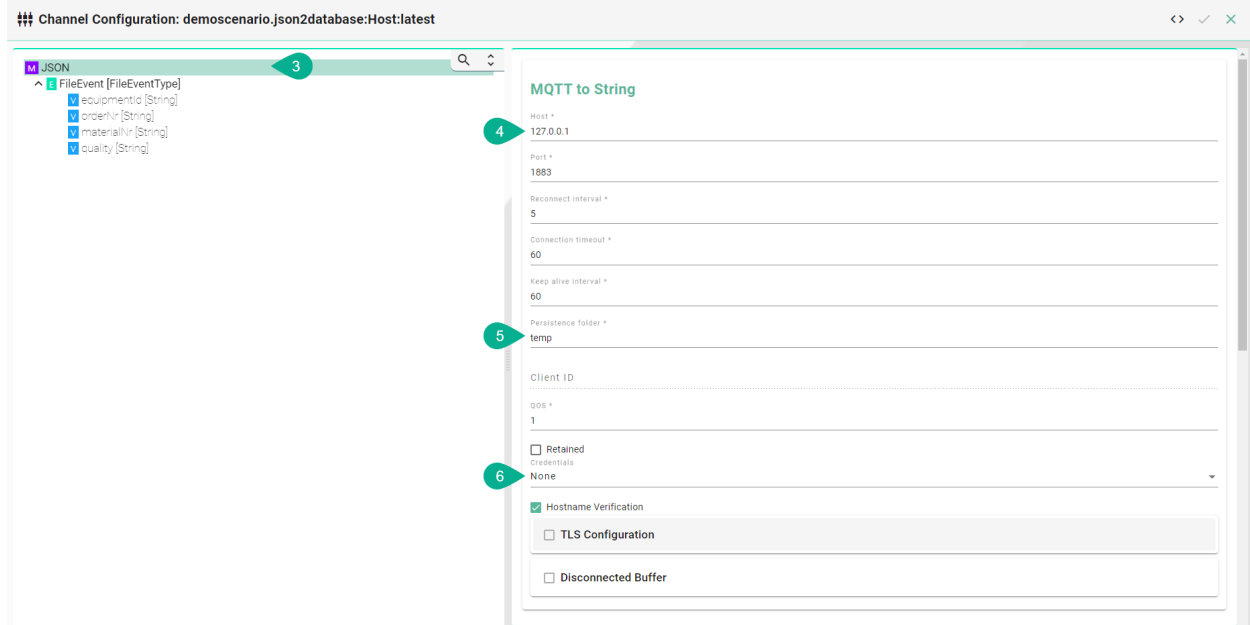
- The first Node after the root node **M** must be of type *Event* **E**.
- The following Node Types can be used under the Event Node:
 - *Variables* **V** with a *Simple Data Type* represents the key-value pairs.
 - *Variables* **V** with a *Custom Data Type* represent objects that can contain key-value 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

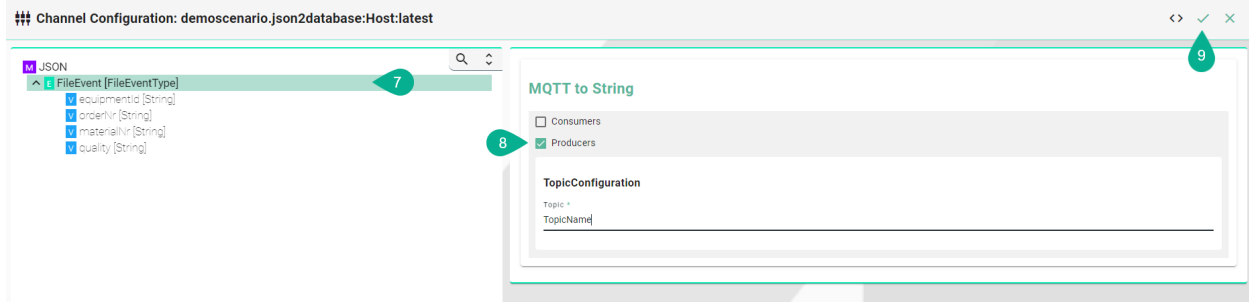
1. Select the **MQTT (JSON)** as Channel Type.
2. Click the **Configure** button.



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. Select **Username and Password** in order to enter **username** and **password**. If there are no credentials needed (e.g., test.mosquitto.org) make sure **None** is selected.



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.



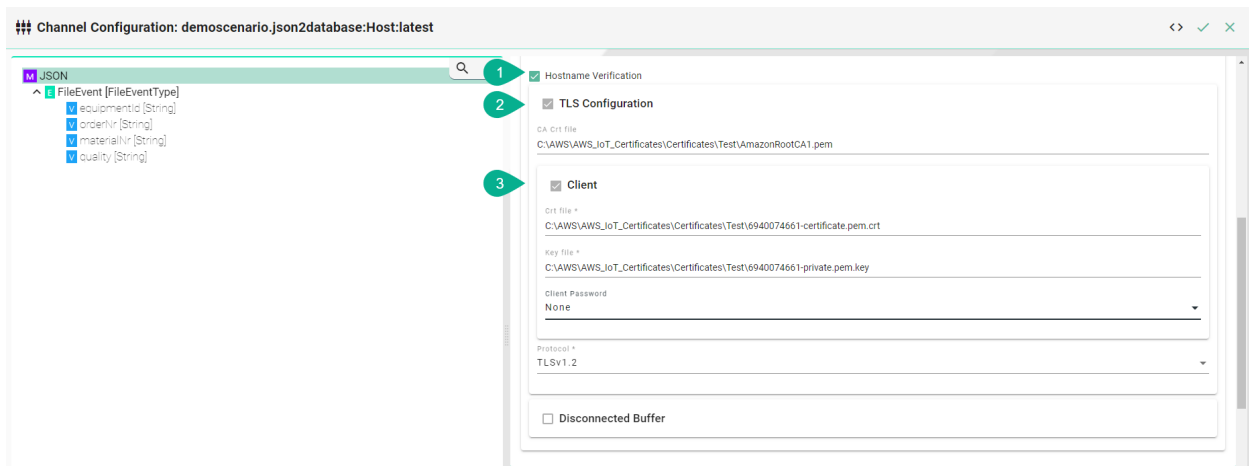
Certificates

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

1. Enable **Hostname Verification** (optional)
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.

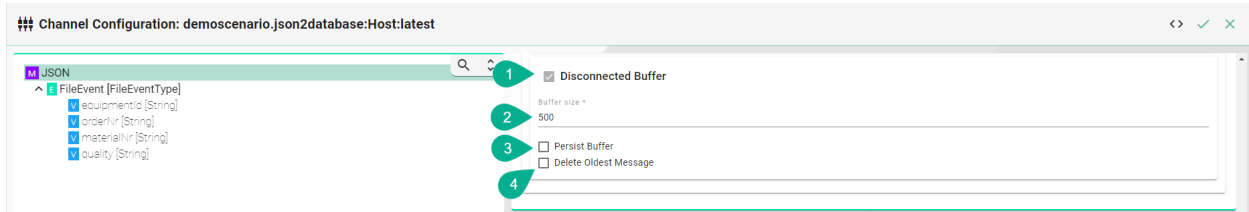
3. Enable the **Client** checkbox
 - Enter the **path** to the **client certificate**. The client certificate identifies the client just like the server certificate identifies the server.
 - Enter the **path** to the **private client key**.
 - If applicable select and enter a **password**.
 - Select 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

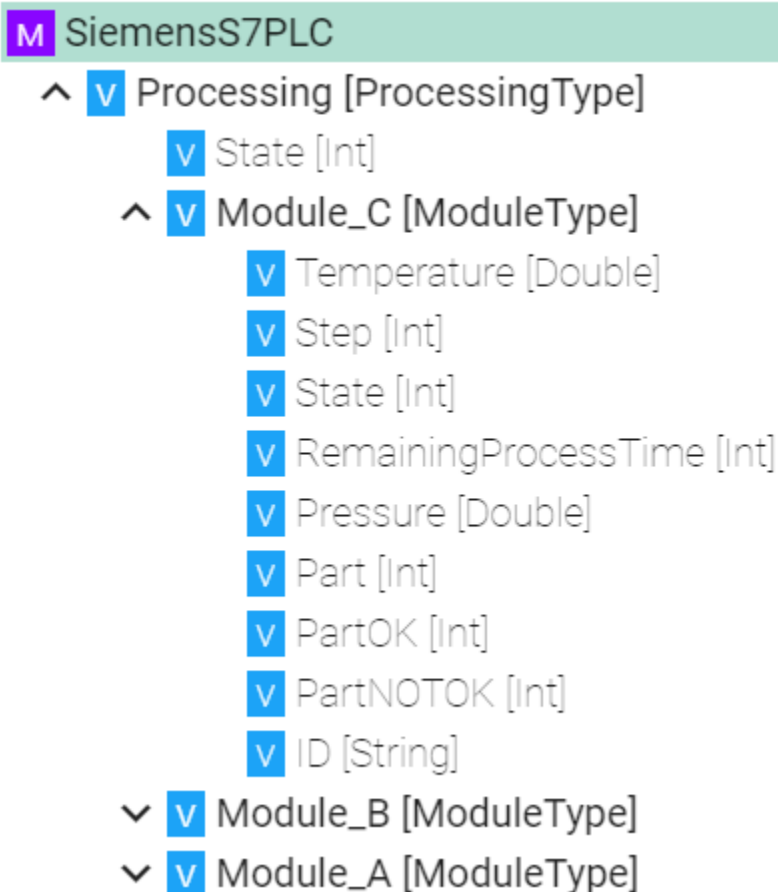
OPC-UA

Characteristics - OPC-UA

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](#).

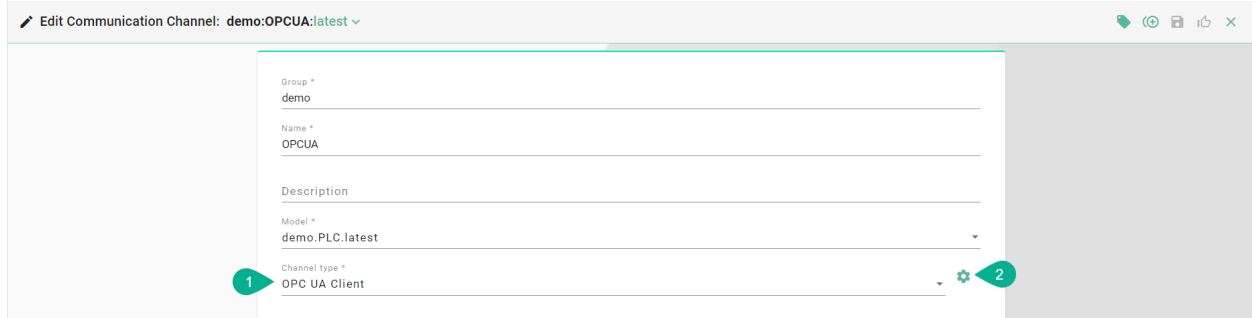
Information Model Requirements

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

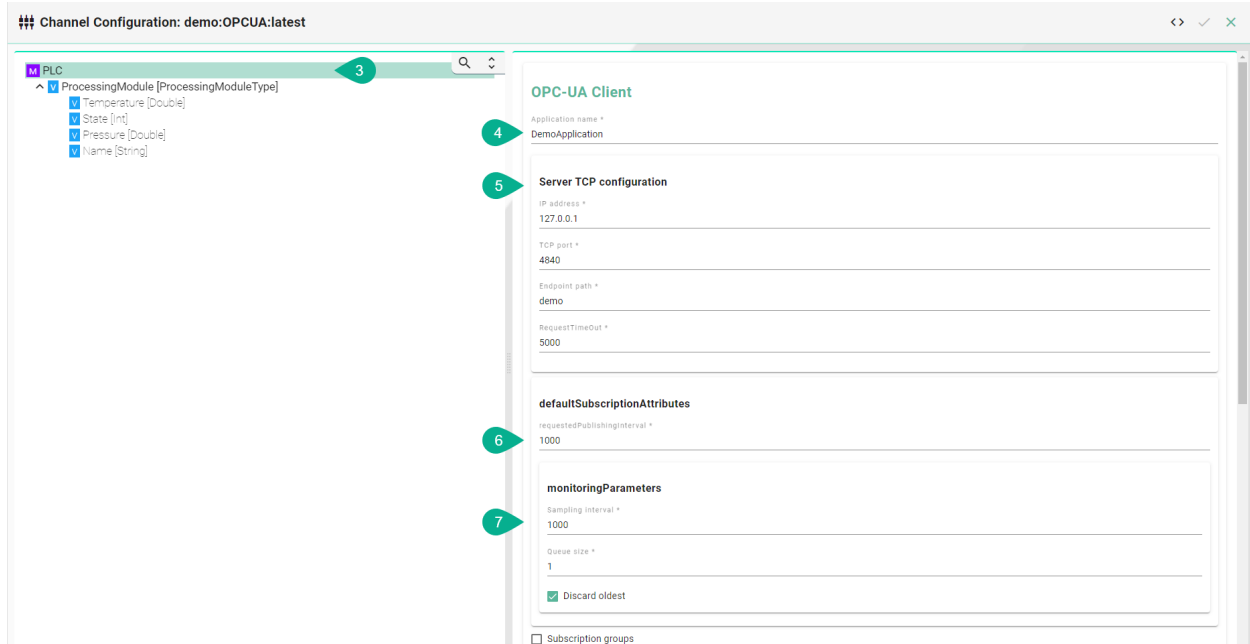


OPC-UA Client

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

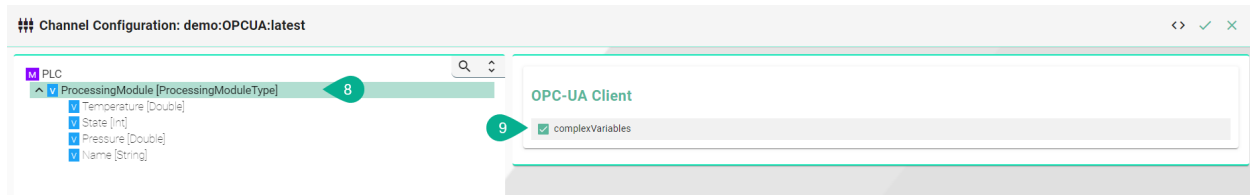


1. **Make sure** the root model node is selected to configure the OPC-UA Client
2. Enter an **applicationName**
3. Configure the serverTcpConfiguration
 - Enter an **ipAddress**
 - Enter the **port**
 - Define an **endpoint**
 - Set a **requestTimeout**
6. Configure the defaultSubscriptionAttribute
 - Define a **publishingInterval**
7. Configure monitoringParameters
 - Set a **samplingInterval**
 - Enter a **queueSize**
 - Enable **discardOldest** depending on the use case



8. Select the complex variable node.

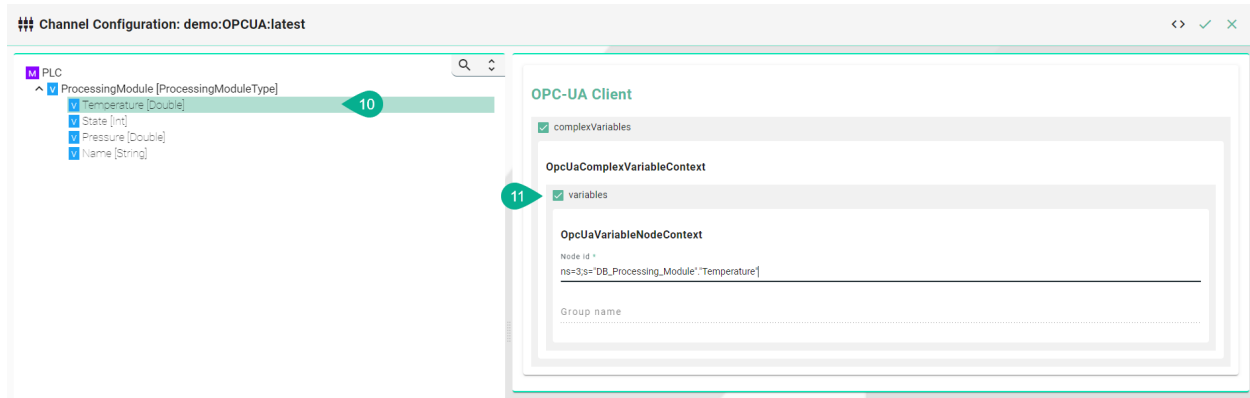
9. Enable the complexVariables



10. Assign OPC-UA data block variables to corresponding variables in the Information Model by selecting the variable in the tree

11. Assign data block

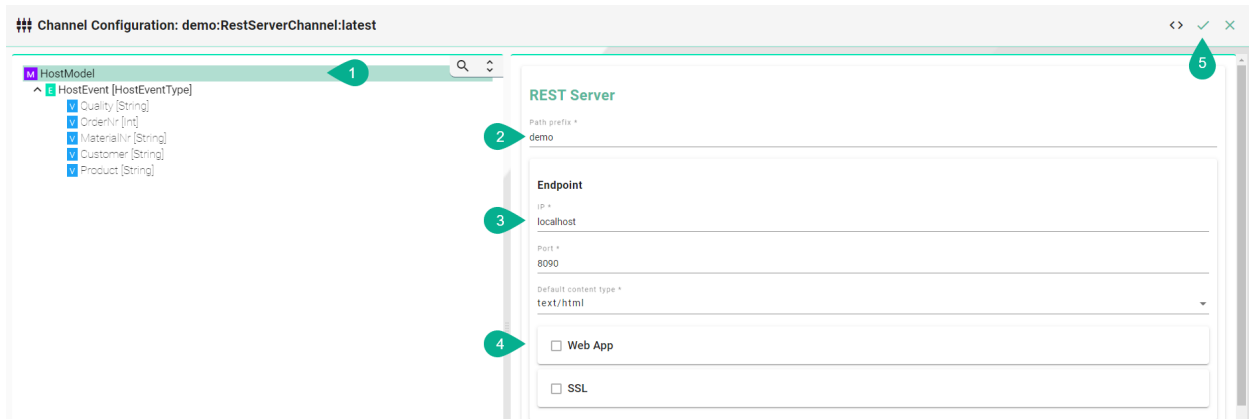
- Enable the **variables** checkbox
- Enter the **nodeId**



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.

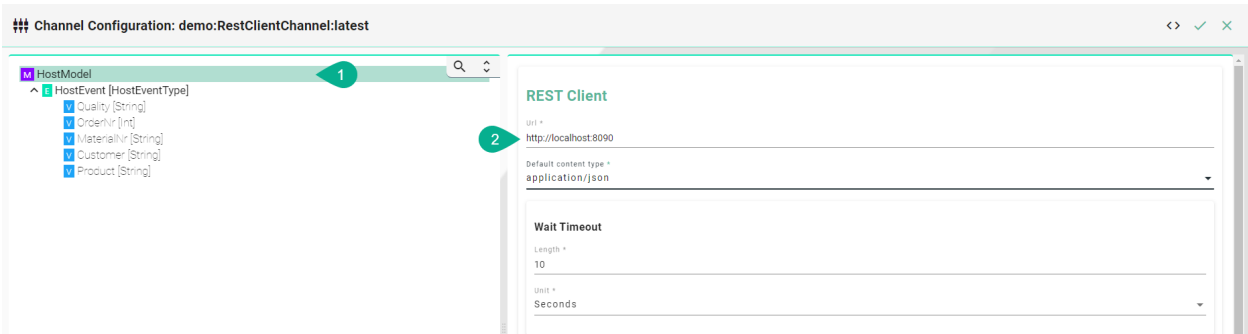


Description of configuration properties:

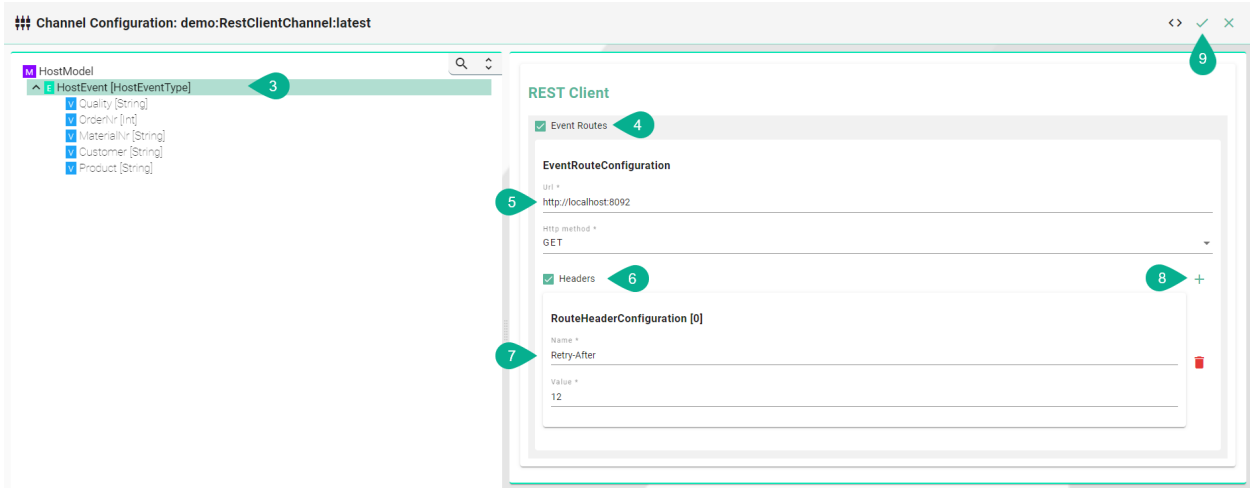
Property	Description	Example
pathPrefix	Prefix for the URL	e.g., demo
Port	Port of the REST server	e.g., 9002, 9000, ...
IP	IP address of the REST server	http://localhost
DefaultContent-type	Is used to indicate the media type of the resource	application/json, application/xml, text/html, text/csv
webapp	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. Enable the **Event routes** checkbox.
5. Variable configuration.
 - Enter the **URL**.
 - Select the **HTTP method**.
6. Enable the **Headers** checkbox to add headers.
7. Configure Key-Value pair of the header.
 - Enter a **name**.
 - Enter a **value**.
8. Click on the **Add header** button to add more headers.
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

2.2.4 General Configurations

These configurations apply for all Communication Channel Types.

Framework Configuration

The Framework Configuration enables insights into data handled by *Mapping Rules*. If enabled, logs will be generated once Rules are triggered and executed. These logs are visible then by default in the **INFO Log Level** as well as in the *Log Viewer*.

The following Framework Logging Configurations are available:

- Stateful Variable
- Stateless Variable

- *Event*
- Command

For each configuration there are two ways to use logging:

- **Enable:** Logs out information about the *Node Type* that was executed by the Rule.
- **Log Data:** Logs out in JSON-format the actual data of the *Node Type* that was executed by a Rule.

FrameworkConfiguration

Logging

Stateful variable

☐ Enable
 ☐ Log Data

Stateless variable

☐ Enable
 ☐ Log Data

Event

☐ Enable
 ☐ Log Data

Command

☐ Enable
 ☐ Log Data

Event Logging

To use the Event Logging enable the checkbox **EventLogging** and for more detailed logging **EventDataLogging**.

EventLogging

☒ EnableLogging
 ☒ EnableDataLogging

Event Logging Output

```
[INFO ] - EventDefinition - Received Event: /Model/bcdbbfd3-cdbe-4ade-8a73-
↪3788e6815c46/Event/ReleaseOrder
```

Event Data Logging Output

```
[INFO ] - EventDefinition - Received Event: /Model/bcdbbfd3-cdbe-4ade-8a73-  
↪3788e6815c46/Event/ReleaseOrder={"Quantity":10,"ProductNumber":"Mv5","OrderNumber":  
↪"Ord154","EquipmentId":"4-SWC2"}
```

2.3 Mappings

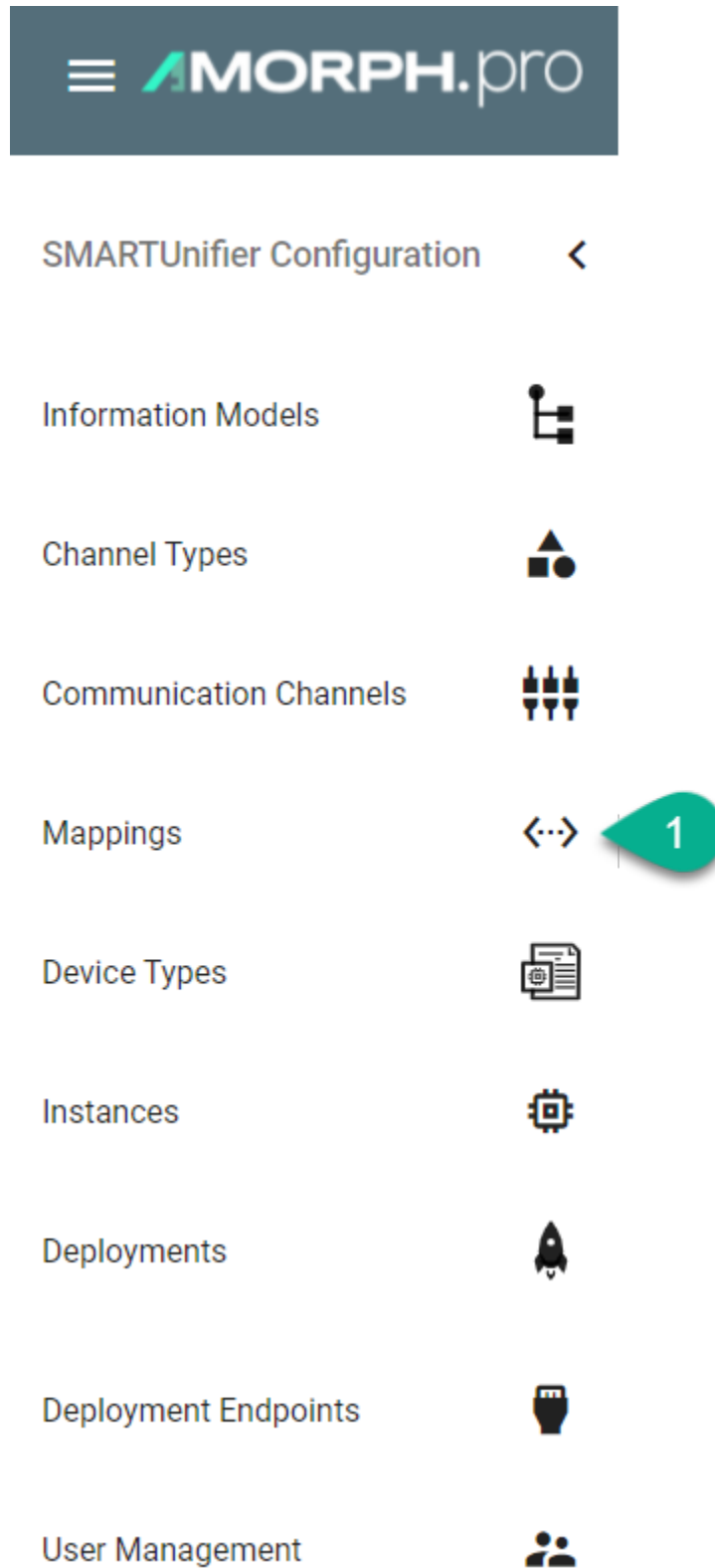
2.3.1 What are Mappings

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

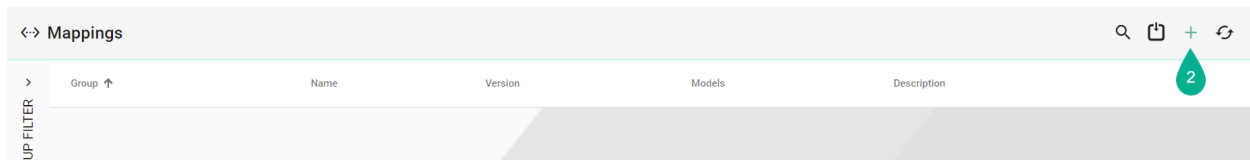
Follow the steps below to create a new Mapping definition:

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

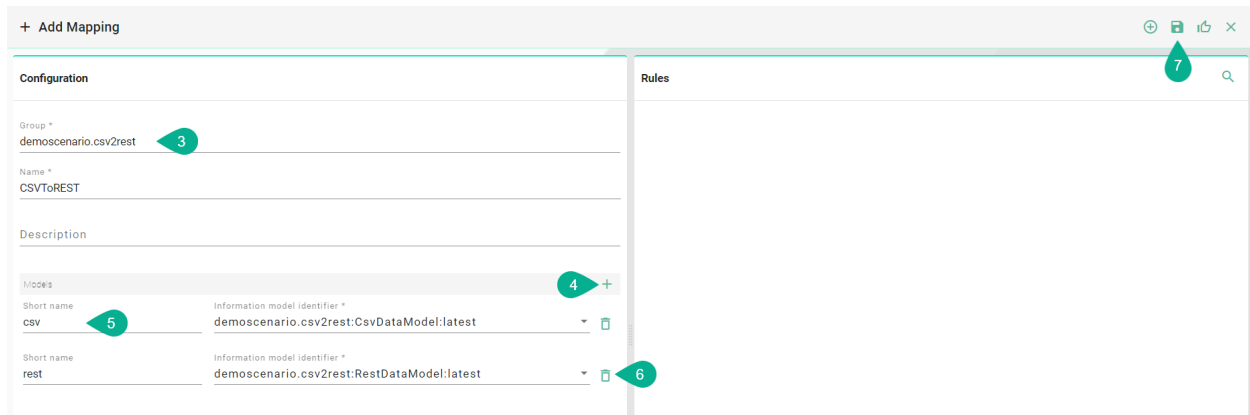


- Following screen containing a list view of existing Mappings is displayed

- 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 *Mapping* (7)
- The newly created Mapping is now visible in the list view

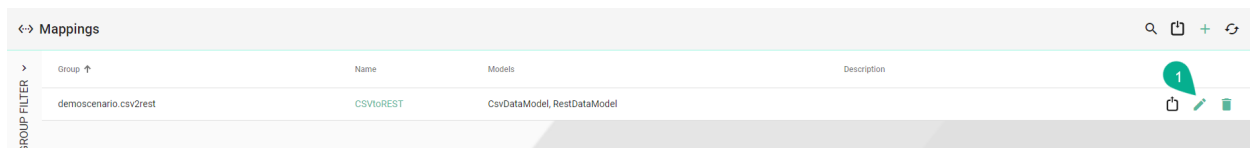


2.3.3 How to create Rules

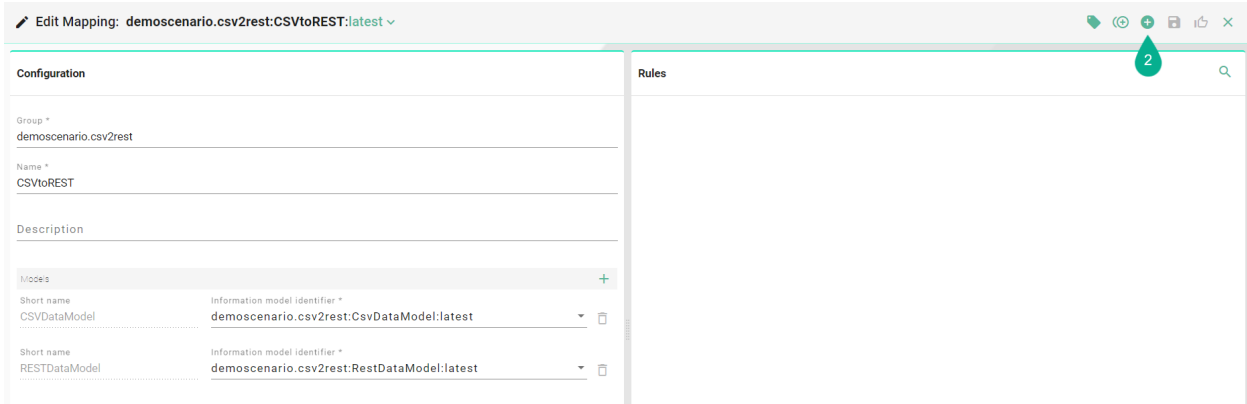
Graphical

Follow the steps described below to create *Rules*:

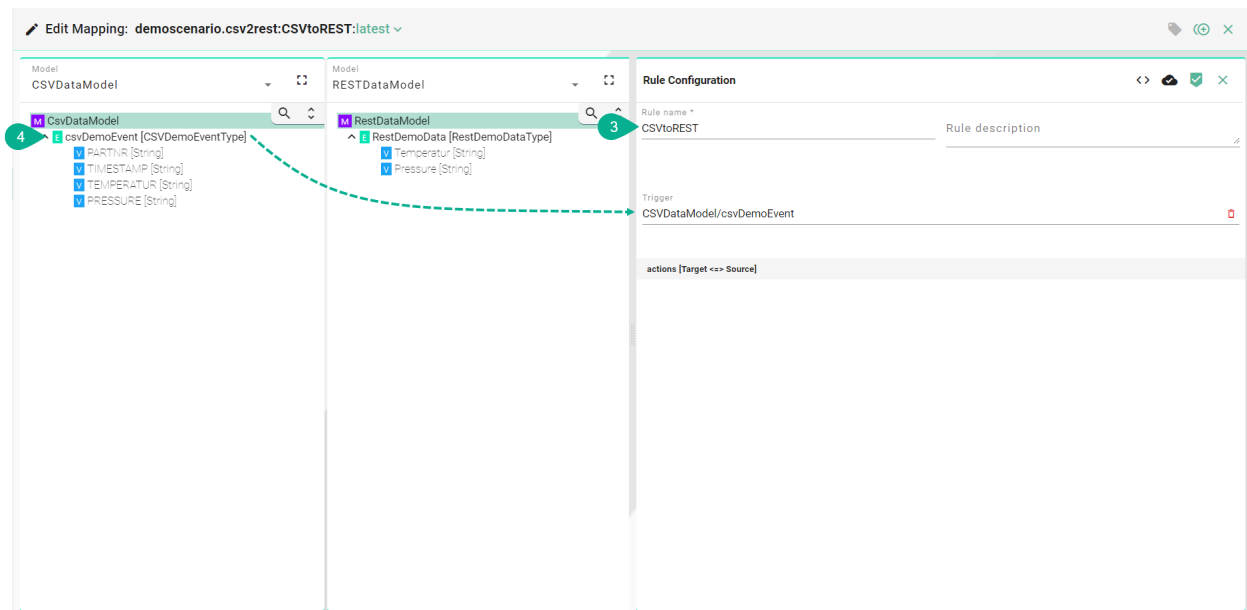
- Select the “Edit” button (1) placed in line with the mapping entry for which the mapping rule is to be created.



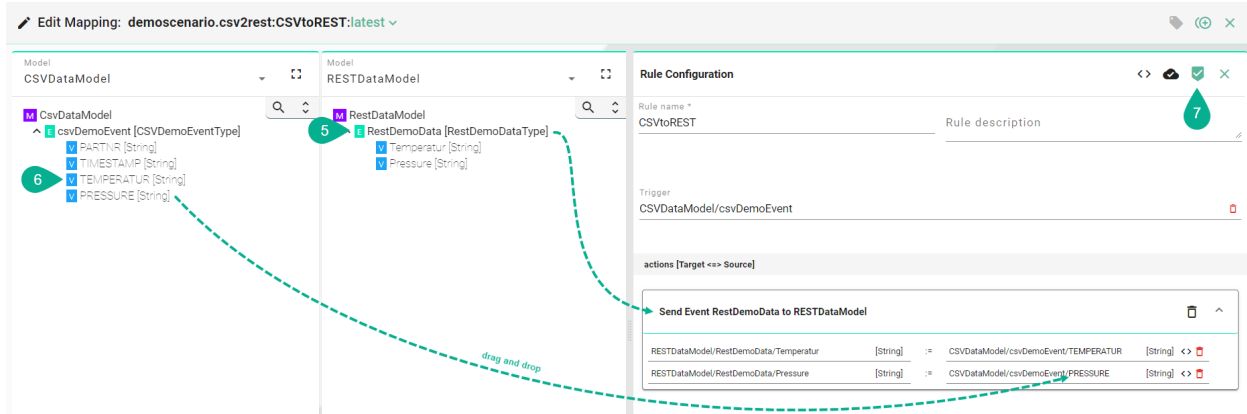
- The edit mapping view is displayed (see figure below):
- Select the “Add Rule” button at the top right corner (2).



- 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.g., String to String)
- After all mandatory fields have been filled out, select the “Apply” button (7) 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
Variable	Command	Variable
		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
Array of a custom type	Event	Variable
		Variable of a custom type
	Command	Variable
		Variable of a custom type
	Variable	Variable
		Variable of a custom type

continues on next page

Table 3 – continued from previous page

Trigger	Target	Source
	Event	Variable of a custom type
		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
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

continues on next page

Table 3 – continued from previous page

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

Code-based Rules

More complex scenarios, which are currently not supported by the graphical view can be implemented via the code editor in the [Scala](#) programming language. Similar to Mappings via drag and drop, there is no knowledge of the underlying communication protocol (e.g., MQTT, OPCUA, etc.) needed. Protocols are hidden behind the corresponding Information Models.

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 { 4 TriggerInstance =>
2               3
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 := 5 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 4).

Listing 1: Rule - StartOrder - Variable/Event

```

1 EquipmentModel.Alarm mapTo {variable =>
2   MesModel.EquipmentAlarm.send(event => {
3     Try {
4       event.EquipmentId := EnterpriseModel.EquipmentName
5       event.OrderNr := EquipmentModel.CurrentOrder.OrderNr
6       event.MaterialID := EquipmentModel.CurrentMaterialID
7       event.AlarmInfo := EquipmentModel.AlarmInfo
8       CommunicationLogger.log(variable, event)
9     }
10  })
11 }

```

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 3).
- 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 3).
- In this case it is also possible to provide the variable *NewMesOrderFlag* with a Boolean like - **true** (line 6).

Listing 2: Rule - TransferNewOrder - Event/Variable

```

1 MesModel.TransferNewOrder mapTo { event =>
2   Try {
3     EquipmentModel.NewOrder.OrderNr := event.OrderNr
4     EquipmentModel.NewOrder.MaterialNr := event.MaterialNr
5     EquipmentModel.NewOrder.Quantity := event.Quantity
6     EquipmentModel.NewMESOrderFlag := true
7   }
8 }

```

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* 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 4).
- The lines 4-6 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 8).
- 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 11-13) as well as from the Reply (line 14-15) to the *MqttEvent*.

Listing 3: Rule - File2MqttWithDB - Event/Commands

```

1 file.FileEvent mapTo {event =>
2   database.DatabaseCommand.execute(command => {
3     Try {
4       command.orderNr := event.orderNr
5       command.materialNr := event.materialNr
6       CommunicationLogger.log(event, command)
7     }
8   }, reply => {
9     mqtt.MqttEvent.send(event1 => {
10      Try {
11        event1.Quality := event.quality
12        event1.OrderNr := event.orderNr
13        event1.MaterialNr := event.materialNr
14        event1.Customer := reply.customer
15        event1.Product := reply.product
16        CommunicationLogger.log(reply, event1)
17      }
18    })
19  })
20 }
```

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             Try {
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     }
15 }
```

Logging

Logging can be added in the Rule implementation by calling - **CommunicationLogger.log** (line 5)

Listing 5: Rule with Logging

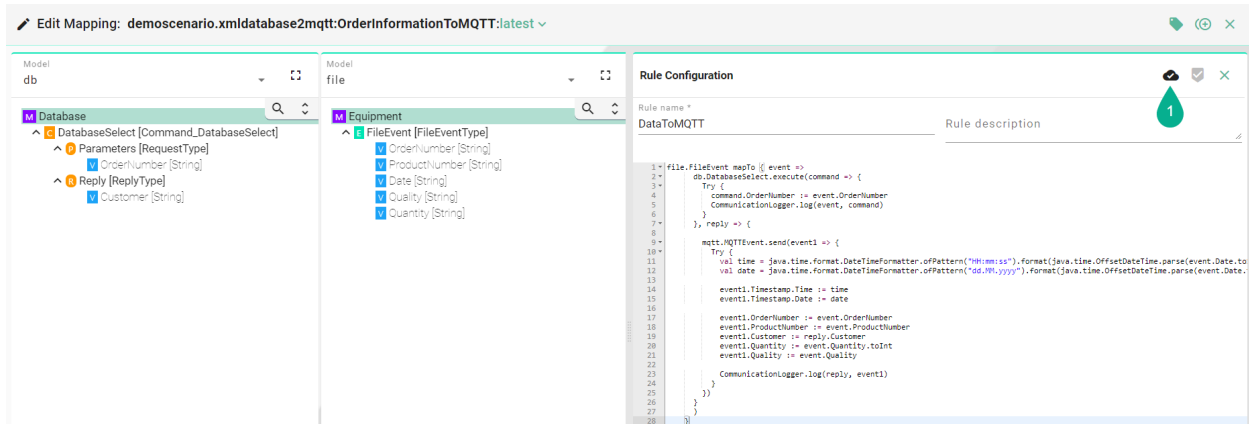
```

1 EquipmentModel.Alarm mapTo {variable =>
2   MesModel.EquipmentAlarm.send(event => {
3     Try {
4       event.EquipmentId := EnterpriseModel.EquipmentName
5       CommunicationLogger.log(variable, event)
6     }
7   })
8 }

```

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

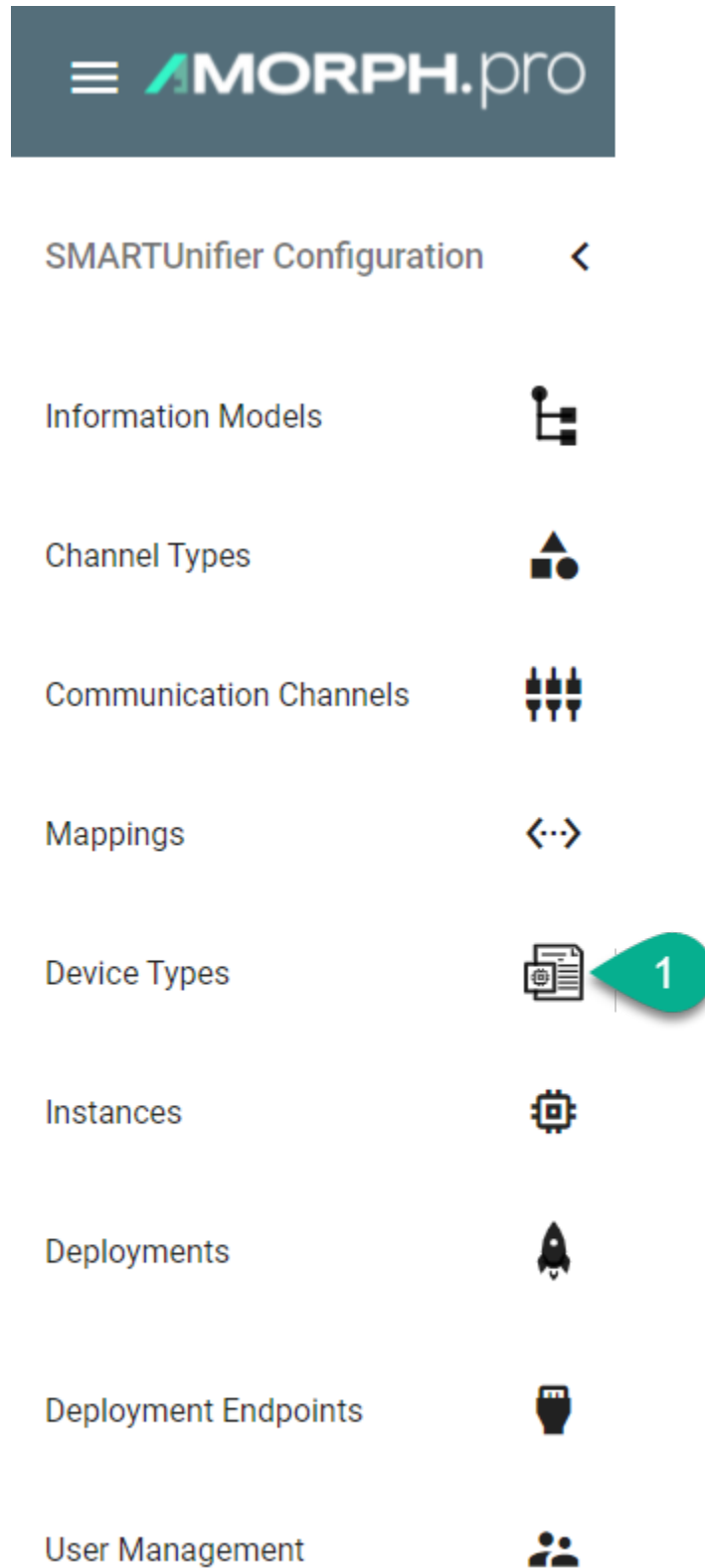
2.4.1 What are Device Types

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).

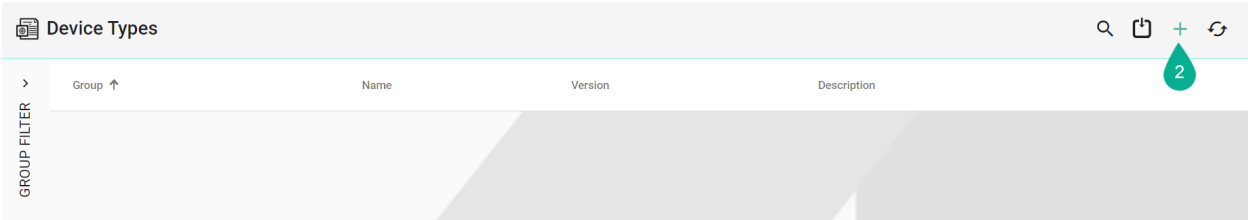
2.4.2 How to create a new Device Type

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

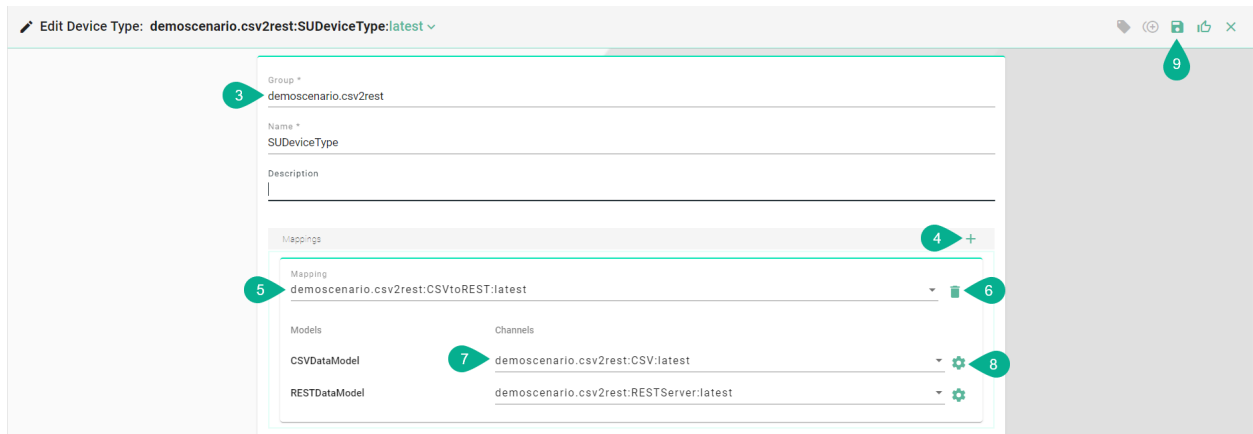
- Select the SMARTUNIFIER Device Type Perspective (1).



- Click on the “Add Device Type” button in 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).



2.5 Communication Instances

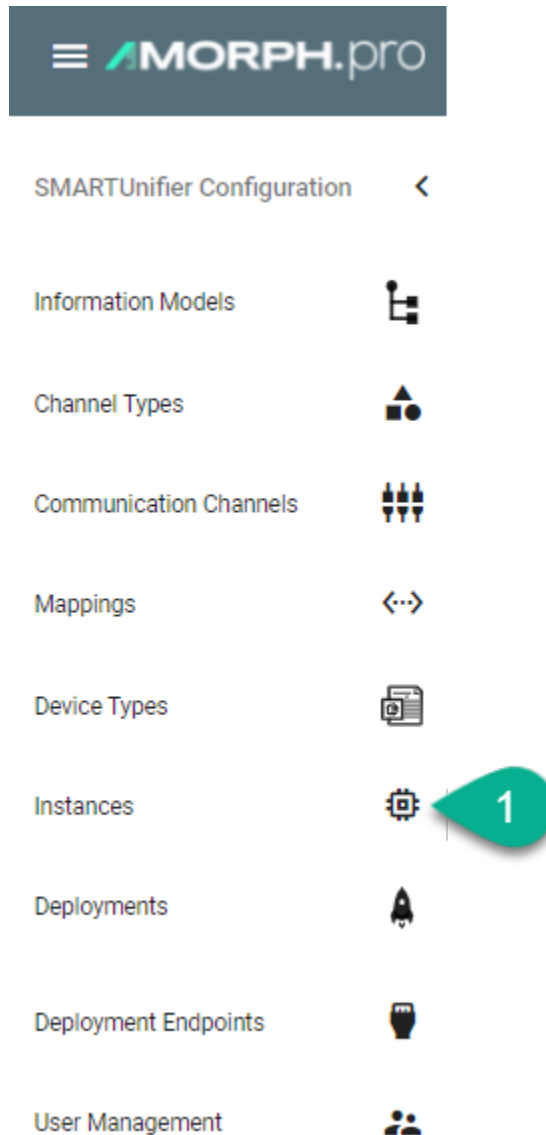
2.5.1 What are Instances

A SMARTUNIFIER Instance is a dynamically created application that can be deployed to any suitable IT resource (e.g., Equipment PC, Server, 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*.

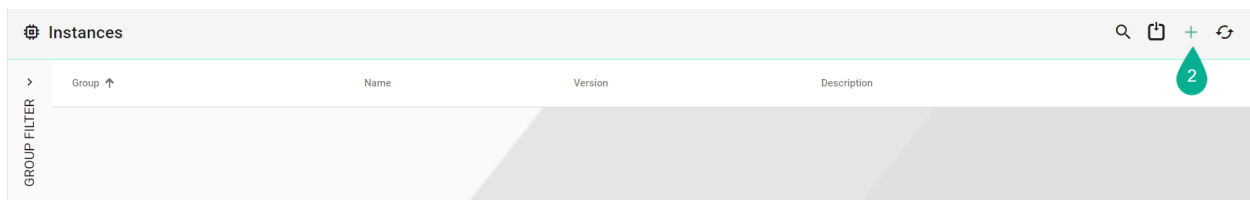
2.5.2 How to create a new Instance

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)

Edit Instance: demoscenario.csv2rest:SUIInstance:latest

Device Type
demoscenario.csv2rest:SUDeviceType:latest

Group *
demoscenario.csv2rest

Name *
SUIInstance

Description

Mappings

Models	Channels
CSVDataModel	demoscenario.csv2rest:CSV:latest
RESTDataModel	demoscenario.csv2rest:RESTServer:latest

Save

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

CONFIGURATION COMPONENT MANAGEMENT

SMARTUNIFIER provides a comprehensive management of the configuration components:

- *Group Filter.*
- *Component Version Control.*
- *Operation.*

In order to keep the SMARTUNIFIER configuration components organized take a look on *how to name the configuration components*.

3.1 Naming Convention

Each *Configuration Component* created with SMARTUNIFIER has defined a Group, a Name, and a Version.

We recommend the following naming convention for better comprehensibility.

Group Identifies the integration scenario across all integration scenarios within the *SMARTUNIFIER Manager*.

Name Is the name of each component, which is part of the integration scenario, such as: *Models, Channels, Mappings, Device Types, Instances* as well as *Deployment Endpoints*.

Version Defines the version of the component - Suggested format: 1.0.0 / 1.0.1 / 2.0.0.

3.2 Group Filter

With the Group Filter it is possible to restrict the number of components according to the substrings in the Group.

The Group Name contains substrings separated by a dot “.”. The Group Filter is then able to visualizes the Group Names in a hierarchical structure.

The *Show All* filter enables the view of all components (1).

Information Models				🔍	📄	+	↻
Group Filter	Group	Name	Description				
<div> <div>Show All</div> <div>demo</div> <div>scenario1</div> <div>demoscenario</div> <div>csv2rest</div> <div>json2database</div> <div>xml2database2mqtt</div> </div>		demoscenario.xml2database2mqtt	Database				
		demoscenario.xml2database2mqtt	Equipment				
		demoscenario.xml2database2mqtt	Host				
		demoscenario.json2database	Database				
		demoscenario.json2database	JSON				
		demoscenario.csv2rest	CsvDataModel				
		demoscenario.csv2rest	RestDataModel				
		demo.scenario1	DatabaseModel				
		demo	DBModel				
		demo	EquipmentModel				

In order to apply a filter, click one of the items in the Group Filter list (2). At the top of the table, the selected filter is visible (3).

Information Models				🔍	📄	+	↻
Filtered by: csv2rest x							
Group Filter	Group	Name	Description				
<div> <div>Show All</div> <div>demo</div> <div>scenario1</div> <div>demoscenario</div> <div>csv2rest</div> <div>json2database</div> <div>xml2database2mqtt</div> </div>		demoscenario.csv2rest	CsvDataModel				
		demoscenario.csv2rest	RestDataModel				

Removing the filter is possible by either clicking the selected item again, selecting the *Show All* option or by clicking the cross in the filter at the top of the table.

3.3 Component Version Control

Component Version Control enables users to version SMARTUNIFIER configuration components such as Information Models, Communication Channels, Mappings, Device Types and Communication Instances.

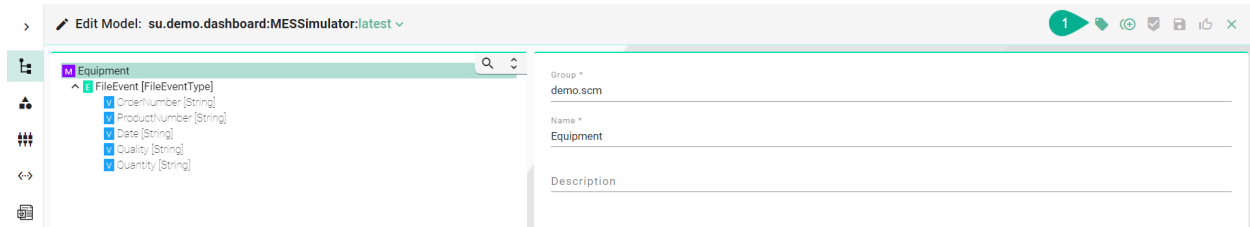
By default, SMARTUNIFIER is using the Component Version Control internally - therefor no configuration is needed. Another option is to point to an external version control system like [Gitea](#). In order to setup an external version control check out the SMARTUNIFIER Installation Guide.

How it works: SMARTUNIFIER creates a repository for each configuration component. Configuration components can be released using tags which reference a specific point in the Git history. After a tag has been created (equivalent to release of a configuration component) there will be no further history of commits/changes. This means that the configuration component can not be edited any further.

3.3.1 How to release configuration components

In order to release a configuration component follow the steps below:

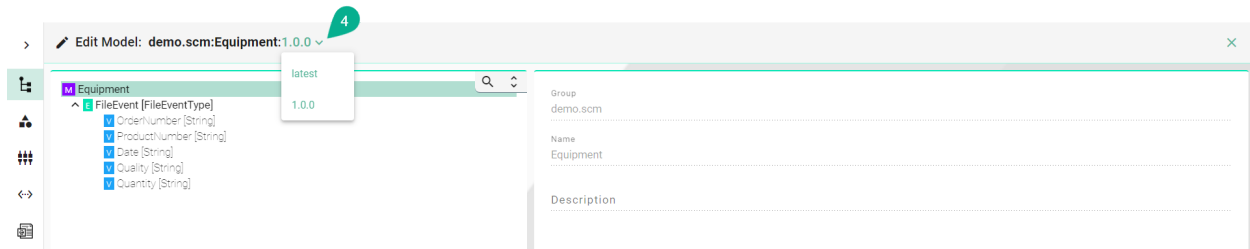
1. Go to an edit page of a configuration component and click the **release** button.



2. Enter an according **version number**.
3. Click **Ok** to confirm.



4. Open the version drop-down to change between **latest** and other **tags**.



Note: Once a configuration component is released you can no longer edit the current tag. If changes are necessary select **latest**. Once you finished editing the final version you can repeat the release process as described above.

3.4 Operations

3.4.1 Add

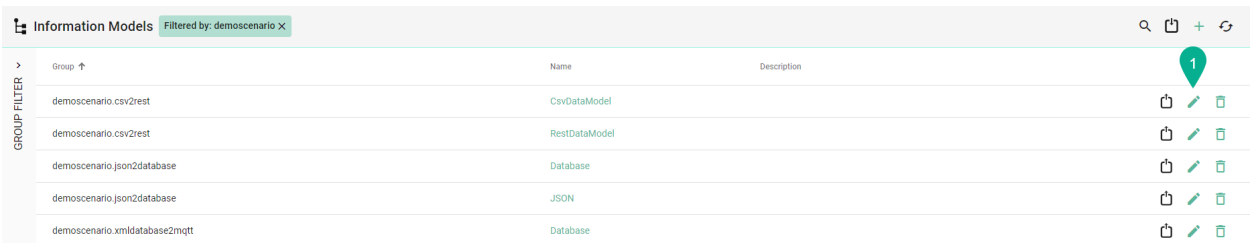
The option to add/create a new component is described in the Instance Setup chapter, for each type:

- *Information Models*
- *Communication Channels*

- *Mappings*
- *Device Types*
- *Instances*
- *Deployments*
- *Deployment Endpoints*

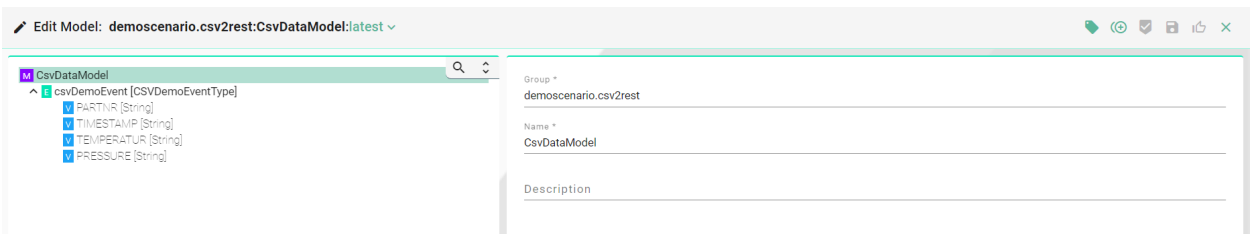
3.4.2 Edit

A component can be edited by clicking the **Edit** button (1).



Information Models		Filtered by: demoscenario x				
	Group ↑	Name	Description			
GROUP FILTER	demoscenario.csv2rest	CsvDataModel		1		
	demoscenario.csv2rest	RestDataModel				
	demoscenario.json2database	Database				
	demoscenario.json2database	JSON				
	demoscenario.xml2database2mqtt	Database				

The component is opened in the edit mode.



Edit Model: demoscenario.csv2rest:CsvDataModel:latest

CsvDataModel

- csvDemoEvent [CSVDemoEventType]
 - PARTNR [String]
 - TIMESTAMP [String]
 - TEMPERATUR [String]
 - PRESSURE [String]

Group *

demoscenario.csv2rest

Name *

CsvDataModel

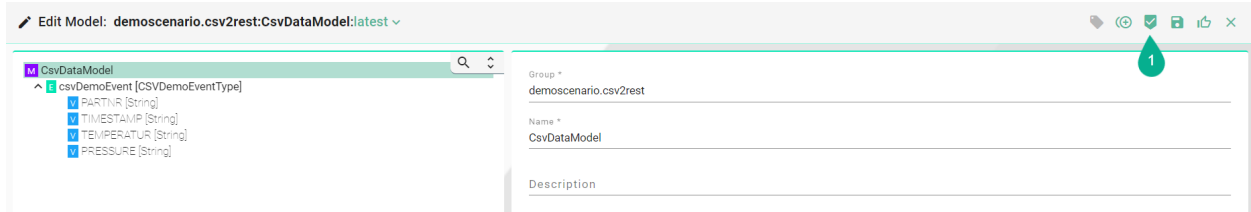
Description

In the edit mode, the following operations are available:

- Clone
- Apply
- Save
- Save and Close
- Close

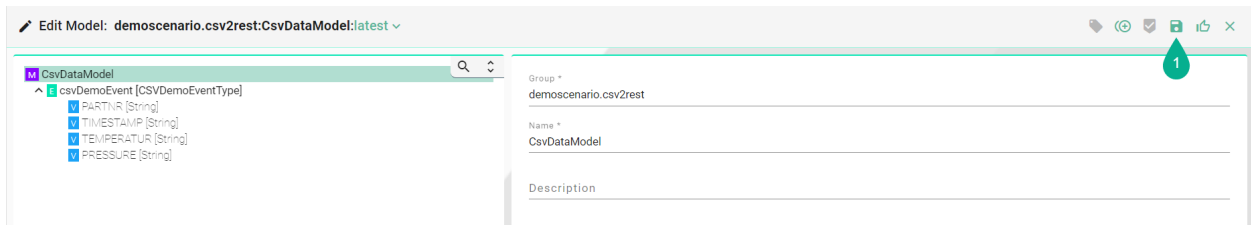
3.4.3 Apply

In the edit mode, after a new data input the **Apply** button (1) must be selected to validate/compile data.

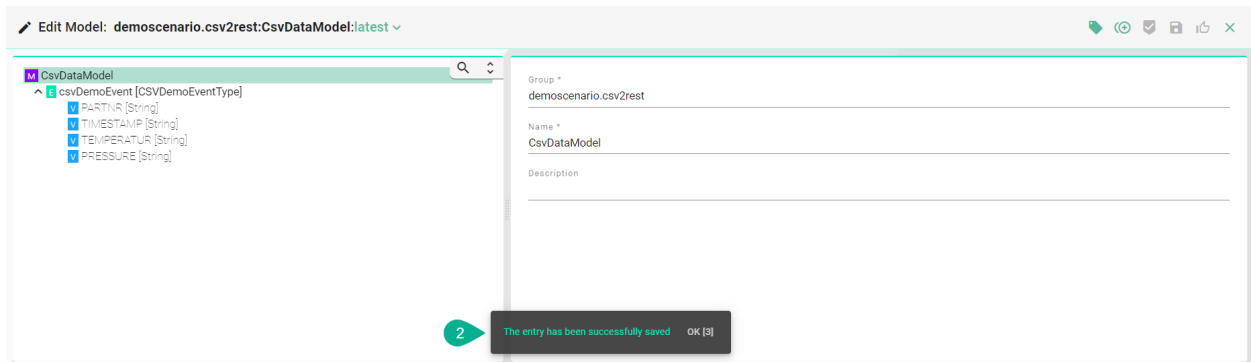


3.4.4 Save

In the Edit Mode, after applying the input data, the user can save the changes by clicking on the **Save** button (1).

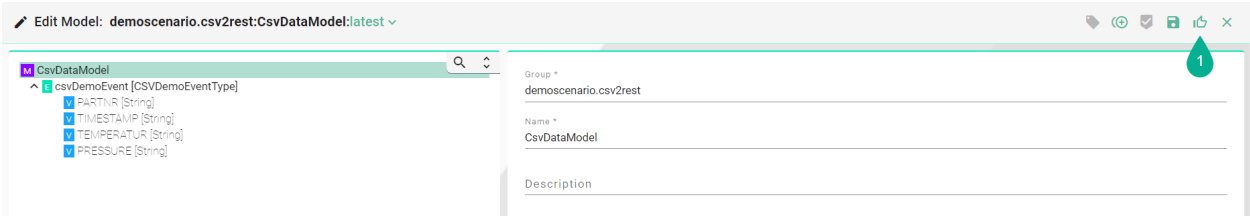


A confirmation message appears (2). The edit mode remains active.

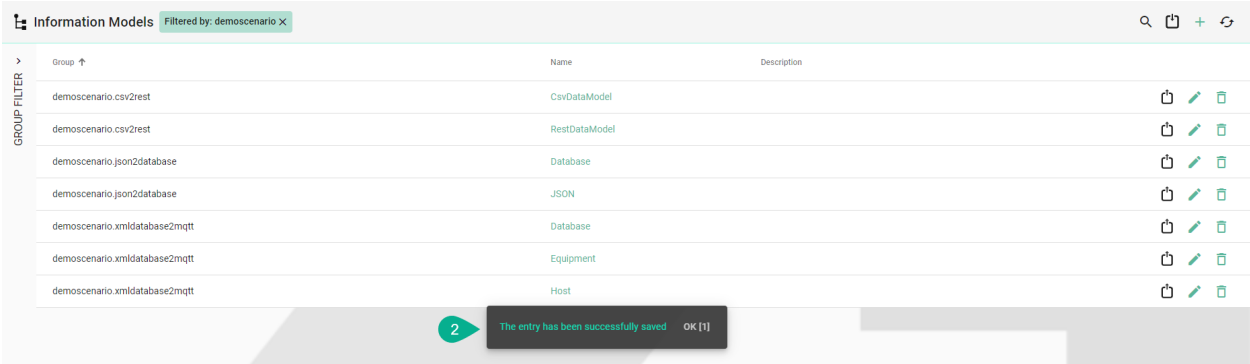


3.4.5 Save and Close

When editing a component, after applying the input data, the user can save the changes and exit the edit mode by clicking on the **Save and Close** button (1).



A confirmation message appears (2). The view mode is active.



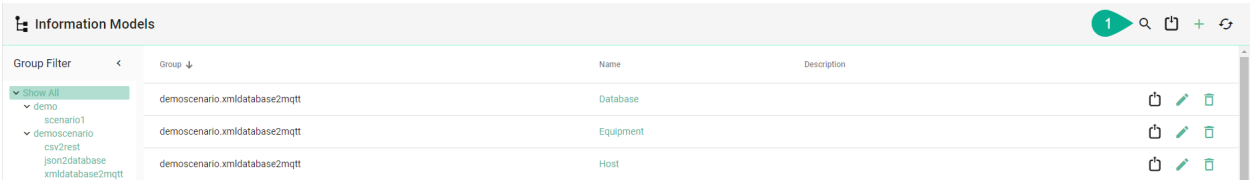
3.4.6 Search

The Search option allows the user to filter results by different criteria:

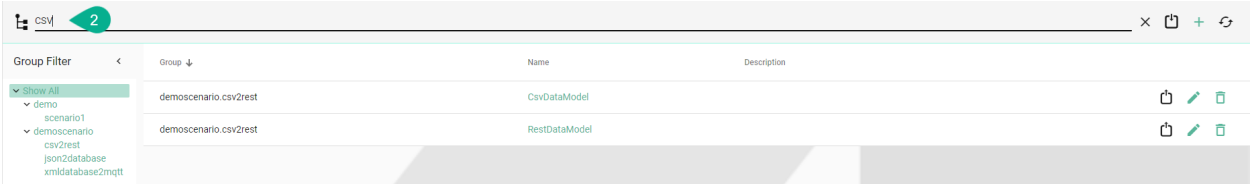
- Name
- Version
- Description

The search is not key sensitive and it works as a partial search, displaying all the results matching with the searched characters.

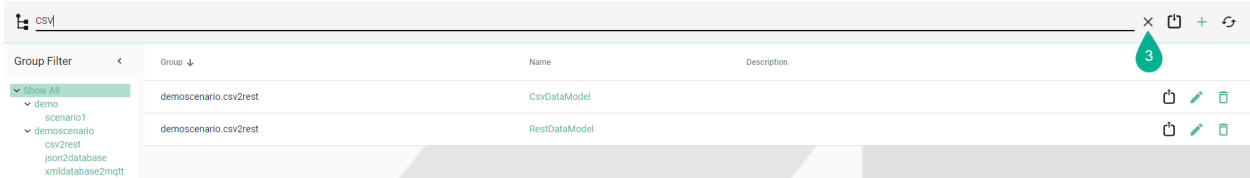
To search for a component, select the **Search** button (1) from the upper right corner.



Enter a search term (2).



To cancel the search click on the **Close Search** button (3).

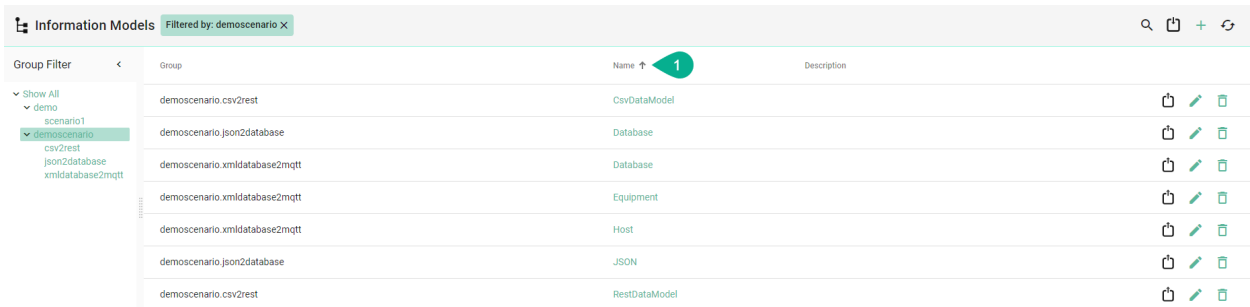


3.4.7 Sort

The information in the view mode can be sorted ascending or descending for each column:

- Group
- Name
- Version
- Description

To sort the information from a column click on the column header (1). An arrow icon will indicate if the components are sorted ascending or descending.

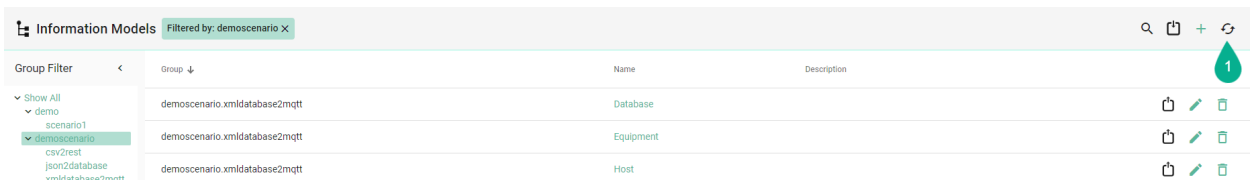


In the view mode on the right of each component, the following operations are available:

- Export
- Edit
- Delete

3.4.8 Reload

This option reloads the components from the repository by selecting the **Reload** button (1) from the upper right.



3.4.9 Import

This option allows the user to add to the scenario a new created or an exported component.

Before importing an exported component, open the JSON file and delete the component **id** (1) - when importing the database will generate a universally unique identifier (uuid). Also, copy (2) and paste (3) the **version** in the **info** section, as shown bellow.



```

1  {
2    "info": {
3      "identifier": {
4        "id": "8b6bc3f2-192d-4870-8bad-b7c2f5e191b9",
5        "version": "latest",
6      },
7      "externalIdentifier": {
8        "name": "CsvDataModel",
9        "group": "demoscenario.csv2rest"
10     },
11     "externalDescriptor": {
12       "description": ""
13     }
14   },
15   "members": [{
16     "id": "csvDemoEvent",
17     "description": "",
18     "definitionType": "Event",
19     "typeName": "CSVDemoEventType"
20   }],
21   "types": {
22     "CSVDemoEventType": {
23       "id": "CSVDemoEventType",
24       "members": [{
25         "id": "PARTNR",
26         "description": "",
27         "definitionType": "Variable",
28         "typeName": "String"
29       }, {
30         "id": "TIMESTAMP",
31         "description": "",
32         "definitionType": "Variable",
33         "typeName": "String"
34       }, {
35         "id": "TEMPERATUR",
36         "description": "",
37         "definitionType": "Variable",
38         "typeName": "String"
39       }, {
40         "id": "PRESSURE",
41         "description": "",
42         "definitionType": "Variable",
43         "typeName": "String"
44       }
45     ]
46   },
47   "category": "model",
48   "type": "StructuredVariable"
49 },
50 "rawModelString": "",
51 "ignoreCompileErrors": false
52 }
53

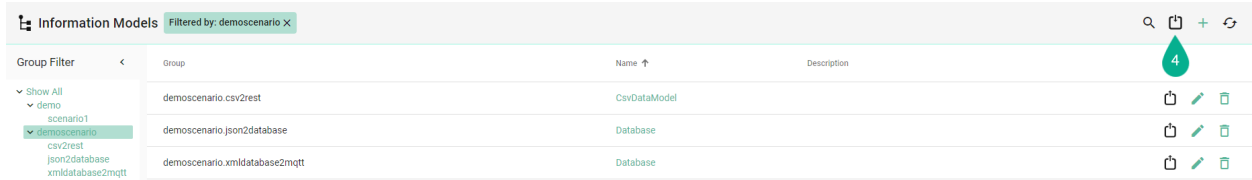
```

```

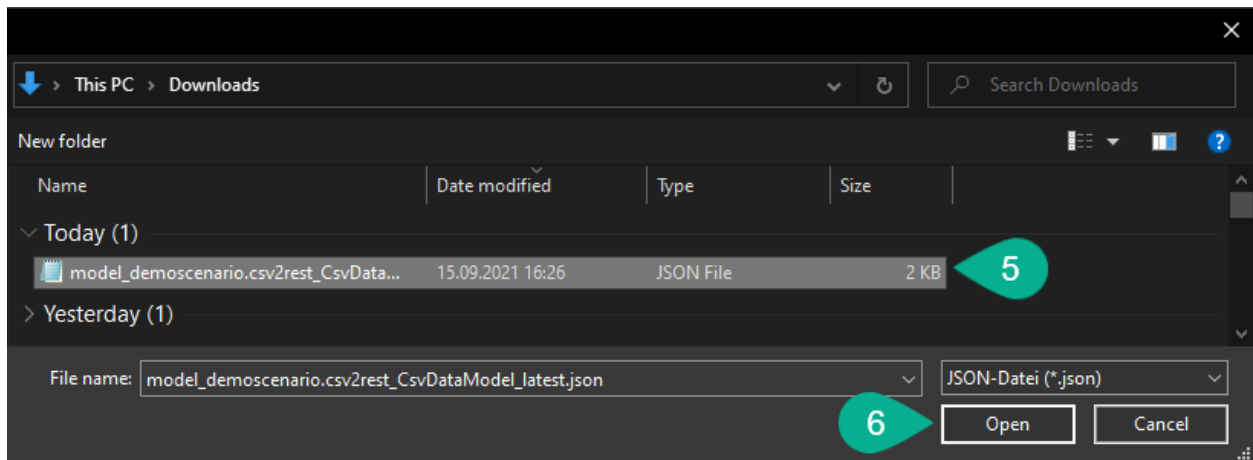
1 {
2   "info": {
3     "identifier": {
4       "id": "",
5       "version": "latest"
6     },
7     "externalIdentifier": {
8       "name": "CsvDataModel2",
9       "group": "demoscenario.csv2rest"
10    },
11    "externalDescriptor": {
12      "description": " "
13    },
14    "version": "latest"
15  },
16  "members": [{
17    "id": "csvDemoEvent",
18    "description": "",
19    "definitionType": "Event",
20    "typeName": "CSVDemoEventType"
21  }],
22  "types": {
23    "CSVDemoEventType": {
24      "id": "CSVDemoEventType",
25      "members": [{
26        "id": "PARTNR",
27        "description": "",
28        "definitionType": "Variable",
29        "typeName": "String"
30      }, {
31        "id": "TIMESTAMP",
32        "description": "",
33        "definitionType": "Variable",
34        "typeName": "String"
35      }, {
36        "id": "TEMPERATUR",
37        "description": "",
38        "definitionType": "Variable",
39        "typeName": "String"
40      }, {
41        "id": "PRESSURE",
42        "description": "",
43        "definitionType": "Variable",
44        "typeName": "String"
45      }
46    ],
47    "category": "model",
48    "type": "StructuredVariable"
49  }
50 },
51 "rawModelString": "",
52 "ignoreCompileErrors": false
53 }
54

```

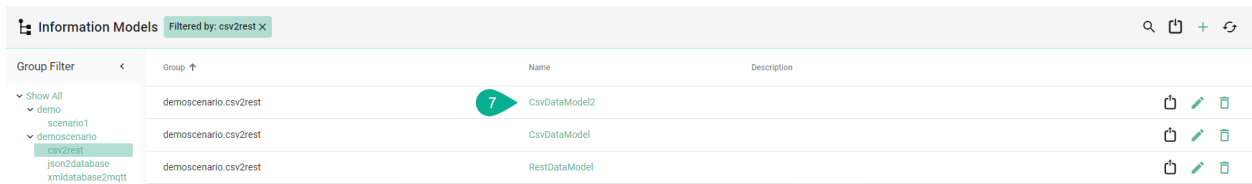
To import, select the **Import** button (4) from the upper right.



A pop-up window appears. Chose the file (5) and select the **Open** button (6).



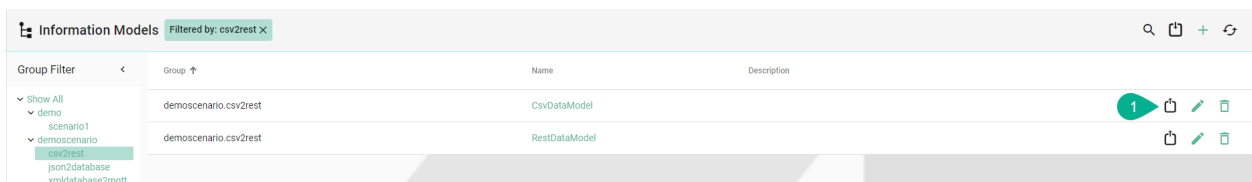
The imported component is now listed (7).



3.4.10 Export

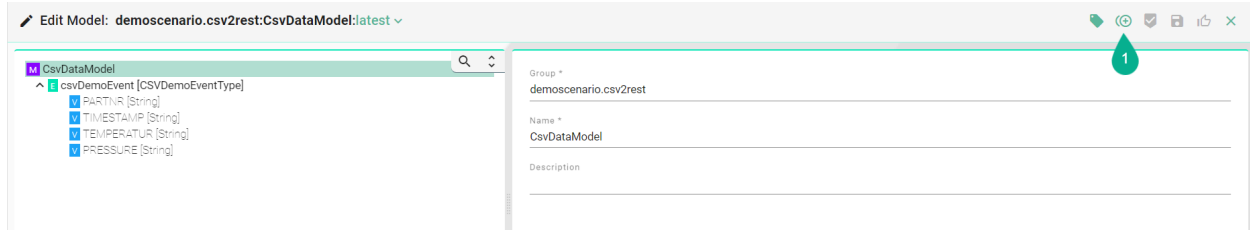
The user has the option to export a component to the local machine.

First, click on the **Export** button (1).



3.4.11 Clone

A component can be cloned from the edit mode, by selecting the **Clone** button. (1).



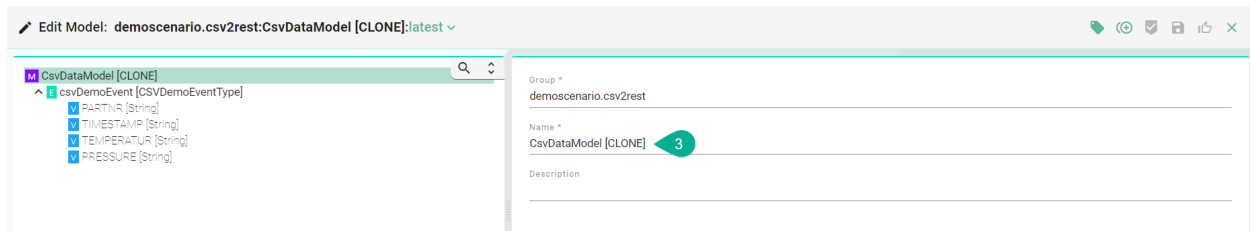
A pop-up appears, click on the **Ok** button (2).

Are you sure you want to clone this configuration component?

A clone of this entry will be created and you will be redirected to the edit for of the new entry



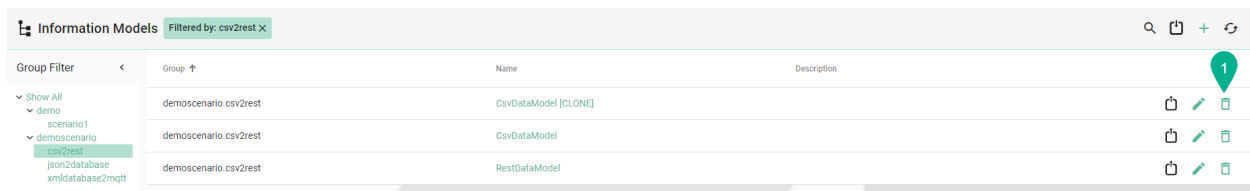
The cloned component is visible, in edit mode, requiring to input a valid name (4)



Note: The Clone operation is not available for the Deployment component.

3.4.12 Delete

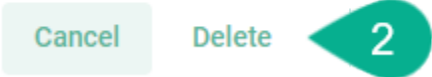
A component can be deleted by clicking the **Delete** button (1).



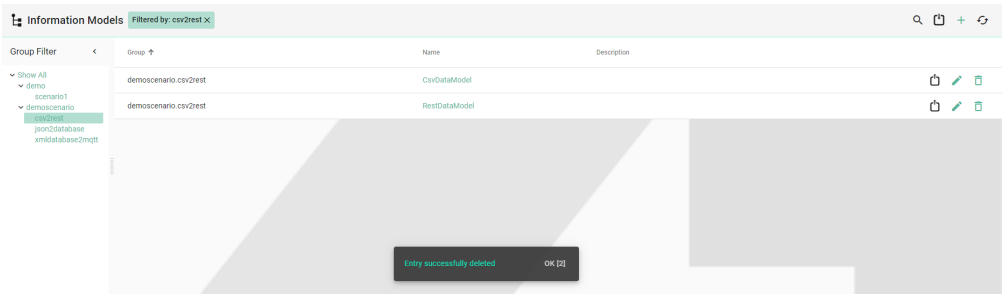
Select **Delete** on the confirmation dialog (2).

Delete Model

Are you sure you want to delete this Information Model?

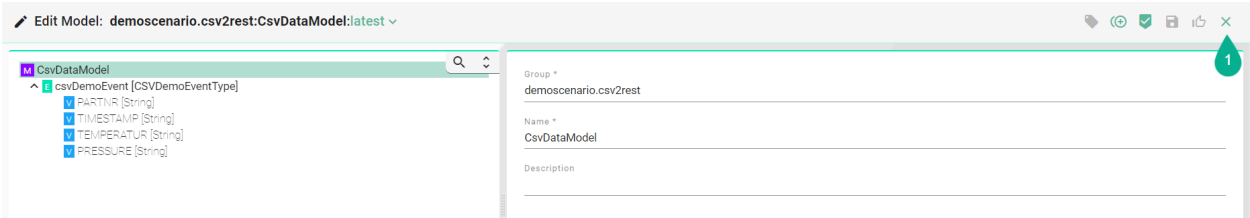


The component is deleted.



3.4.13 Exit Editing

The user can exit the edit mode by clicking on the **Close** button (1).



If the data is not saved, a pop-up appears and the user can select the **Cancel** button (2) to return to the edit mode and save the data or select the **Leave** button (3) to exit without saving.

Unsaved data

There are unsaved changes on this page. Are you sure you want to leave?



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.

4.1 What is a Deployment

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

Depending on the Deployment Type a Deployment Endpoint must 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 Communication Instance to your local computer where the SMARTUNIFIER Manager is running on.
- **Docker:** Deployment of a SMARTUNIFIER Communication Instance in containerized environments.
- **AWS:** Deployment of a SMARTUNIFIER Communication Instance on the AWS Cloud using AWS Fargate.

SMARTUNIFIER Communication Instance can be encrypted prior the deployment by enabling the encryption option. You learn how to do so in the chapters of the specific deployment options.

Getting started:

- Select your environment and create the Deployment:
 - *Local*
 - *Docker*
 - *Fargate*

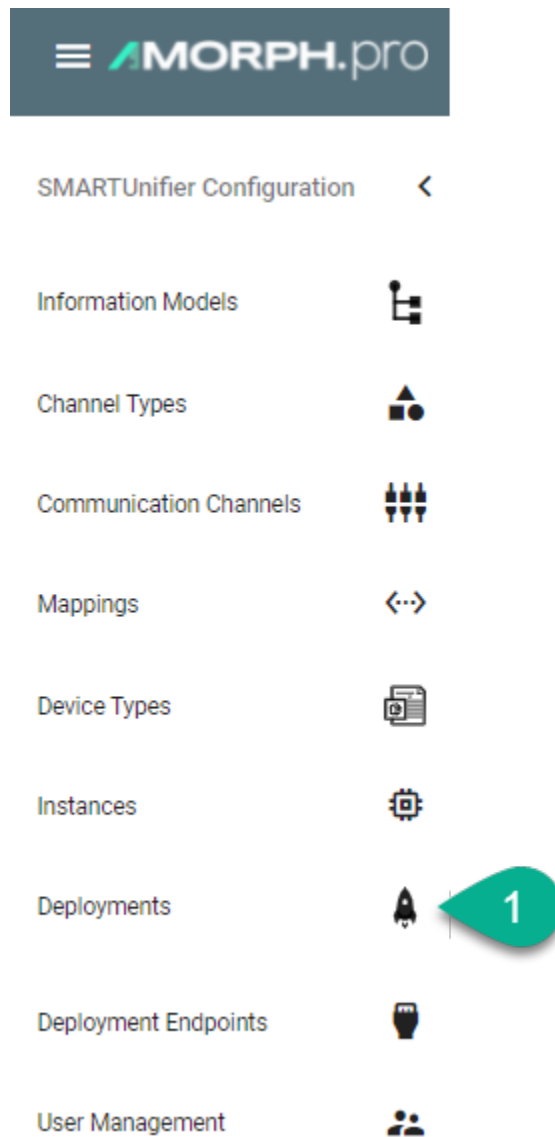
- Learn how to *operate* an Deployment.
- Learn how to *monitor* an Deployment.

4.2 Deploy Locally

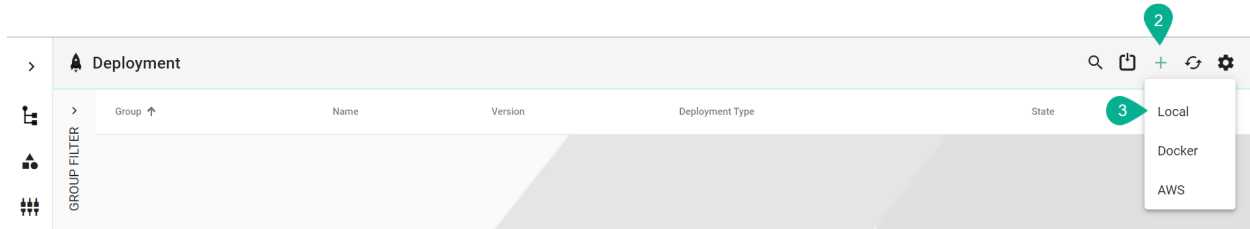
SMARTUNIFIER Communication Instances can be deployed on the IT-resource where the SMARTUNIFIER Manager is running on (e.g., a computer, a server or the AWS Cloud).

Follow the steps described below in order to deploy a Communication Instance locally:

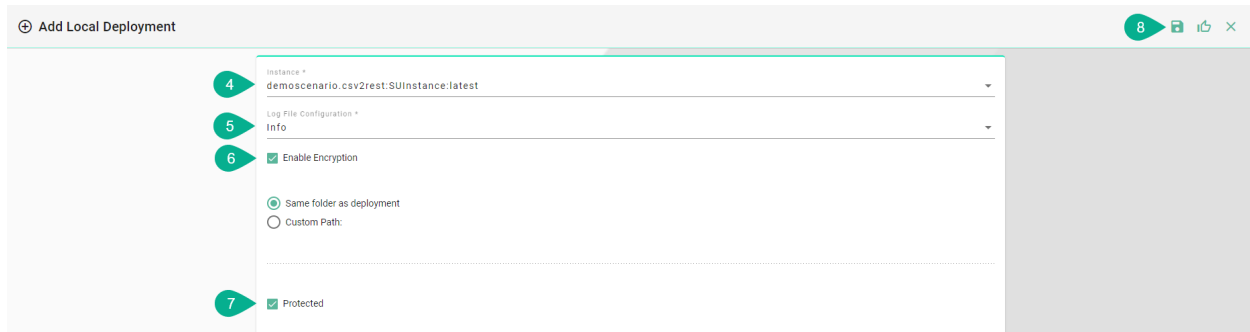
- Select the SMARTUNIFIER Deployment perspective (1).



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



- Select the SMARTUNIFIER Communication Instance to be used in the Deployment (4).
- Select the *log file level* (5). We recommend the log level of type *Info* in case of a normal deployment scenario.
- You can encrypt the SMARTUNIFIER Communication Instance by enabling the encryption checkbox (6).
- Enable “Protected” (7) if you want to secure the deployment - A confirmation will be required for further changes on the deployment (e.g., deploy, undeploy, start, stop).
- When all mandatory fields are filled click the “Save” button (8).



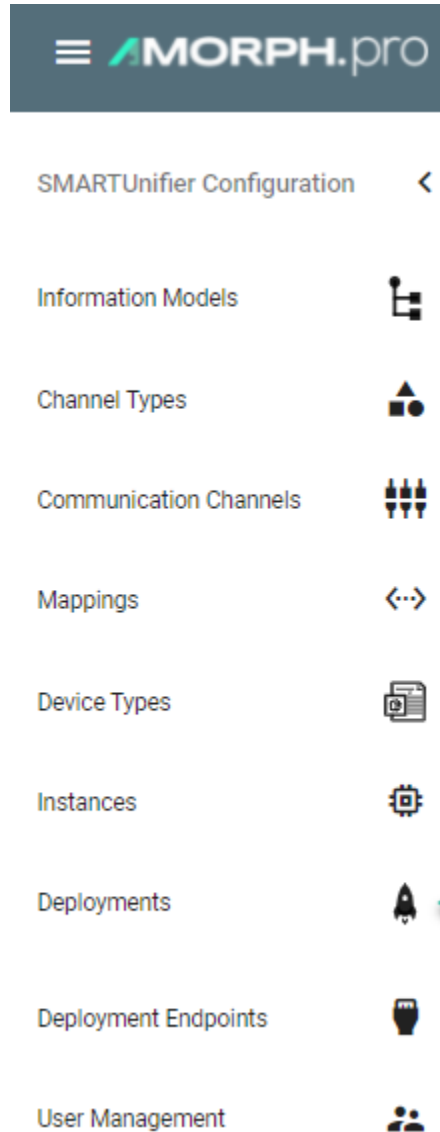
4.3 Deploy with Docker

Note: Before deploying a Communication Instance with Docker make sure to add a *Docker Java Image* and to create a *Docker Deployment Endpoint*. The Deployment Endpoint specifies the location where you want the container to run.

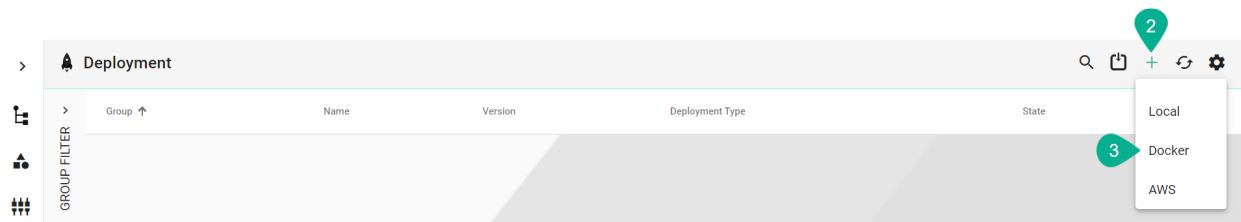
SMARTUNIFIER Communication Instances can be deployed on any location that has an existing Docker environment in place.

Follow the steps described below to deploy a Communication Instance inside 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 Communication 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 added in the [Docker Java Image Manager](#) from the Drop-Down menu (6).

- Select the *log file level* (7). We recommend the log level of type *Info* in case of a normal deployment scenario.
- Enable the encryption checkbox if you want to encrypt the SMARTUNIFIER Communication Instance (8)
- Enable “Protected” (9) if you want to secure the deployment - A confirmation will be required for further changes on the deployment (e.g., deploy, undeploy, start, stop).
- When all mandatory fields are filled click the “Save” button (10).

The screenshot shows a web form titled "Add Docker Deployment". It contains several input fields and checkboxes, each with a numbered callout (4-10) pointing to it. The fields are:

- 4: Instance * (dropdown menu showing "demoscenario.csv2rest:SUInstance:latest")
- 5: Endpoint ID * (dropdown menu showing "demo:DockeEndpoint1")
- 6: Image * (dropdown menu showing "adoptopenjdk:11-jre-hotspot")
- 7: Log File Configuration * (dropdown menu showing "Info")
- 8: Enable Encryption (checkbox)
- 9: Protected (checkbox)
- 10: A green button with a white document icon and a close 'x' icon, representing the 'Save' button.

4.4 Deploy with AWS Fargate

SMARTUNIFIER supports the deployment of Communication Instances on Amazon Web Services (AWS) using AWS Fargate. Using AWS Fargate removes the operational overhead of managing servers by paying only for the resources actually used.

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.

4.4.1 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 **programmatic 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 to interact with dependent AWS services:

- Access to Amazon S3 to retrieve SMARTUNIFIER Instance artifacts - such as libraries and configuration files.
- Access to AWS ECR 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": [
        "*"
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```

    ]
  },
  {
    "Sid": "ECRPullPolicy",
    "Effect": "Allow",
    "Action": [
      "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": "*"
  }
]
}

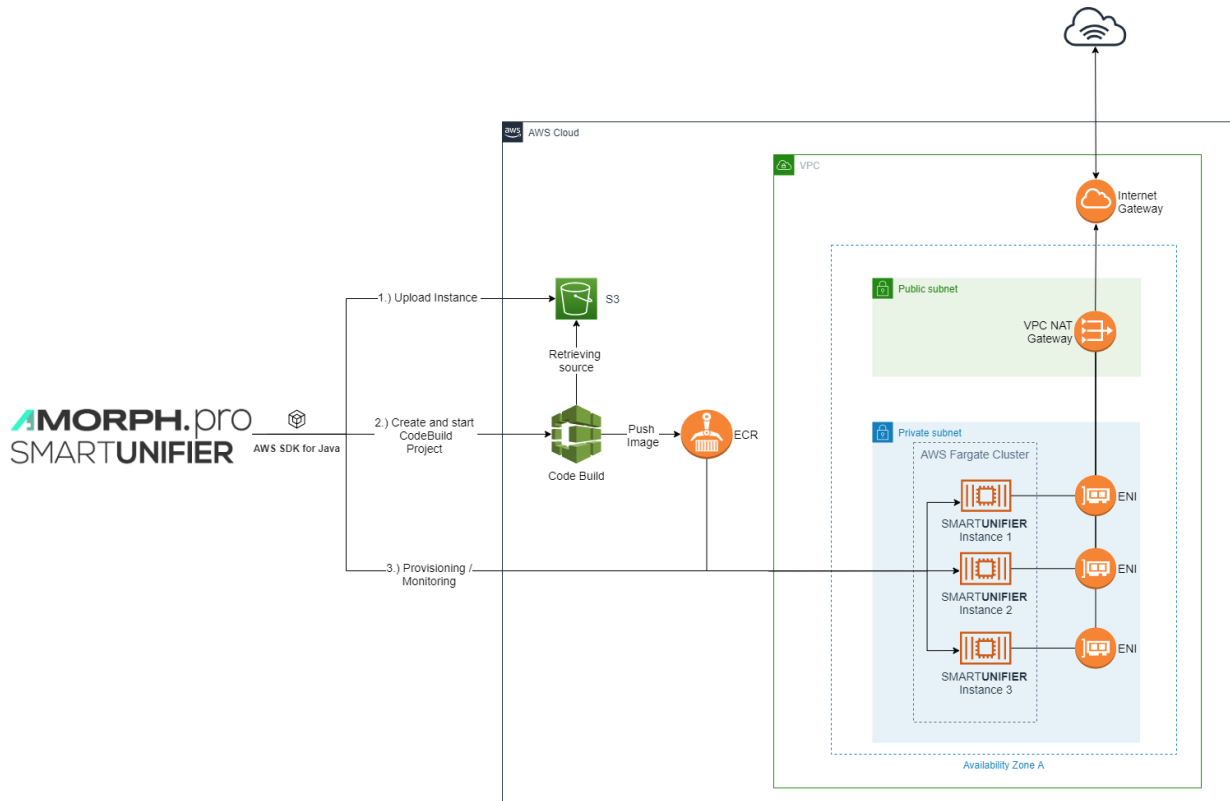
```

4.4.2 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. To run SMARTUNIFIER Manager on AWS Cloud please see the SMARTUNIFIER Installation Manual.

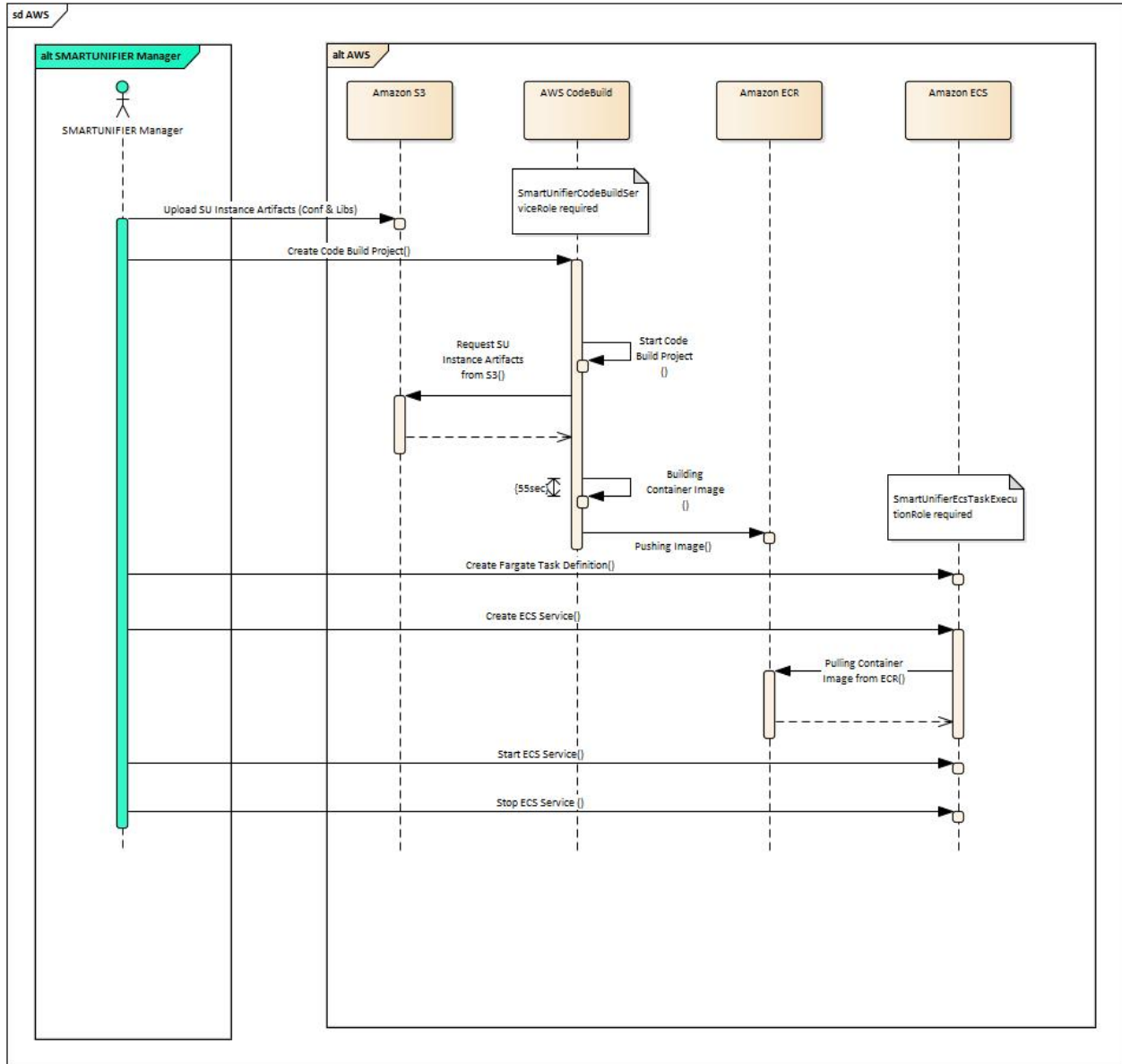
SMARTUNIFIER is using the [AWS SDK for Java](#) to make deployments of Instances to [AWS Fargate](#). Following AWS Services are used during the deployment process:

- 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.



4.4.3 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	

4.4.4 Deployment Steps

Expected Time

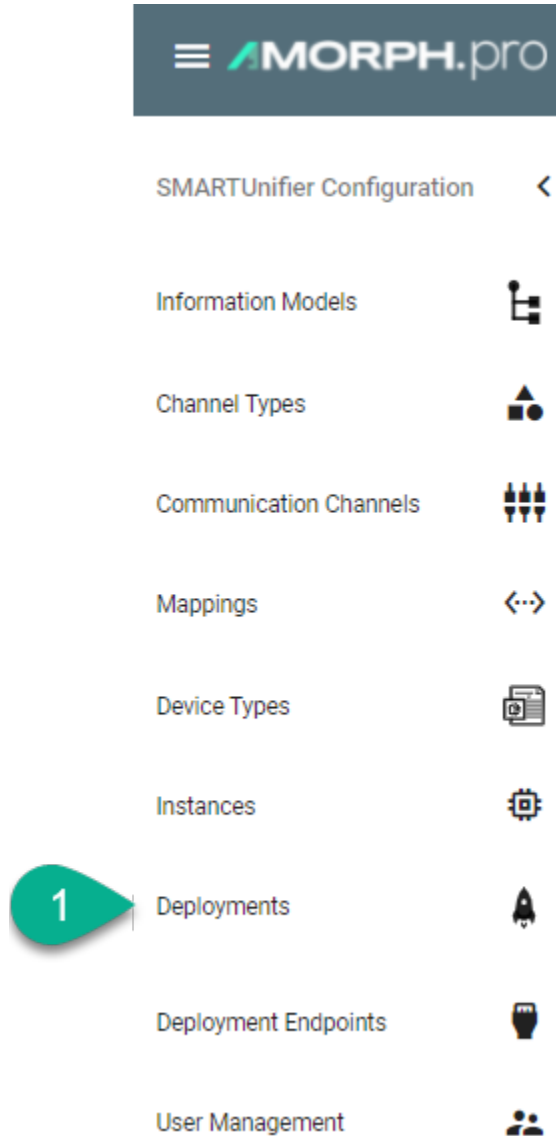
- Deployment of an SMARTUNIFIER Instance on AWS Fargate (Existing AWS Resources) - expected deployment time: **3-5 min**
- Deployment of an SMARTUNIFIER Instance on AWS Fargate (Creation of needed AWS Resources required) - expected deployment time: **20-30 min** (Please note that this is a one time setup of the customers AWS cloud infrastructure)

Deployment of the SMARTUNIFIER Instance

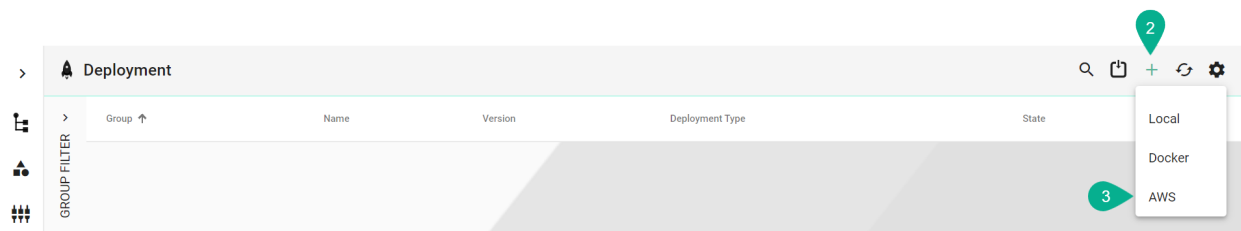
If you have not already set up an AWS Deployment Endpoint please refer to chapter: [AWS Endpoint](#).

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 level* (6).
- Enable the encryption checkbox if you want to encrypt the SMARTUNIFIER Communication Instance (7)
- Enable “Protected” (8) if you want to secure the deployment - A confirmation will be required for further changes on the deployment (e.g., deploy, undeploy, start, stop).
- Save the Deployment by clicking the “Save” button (9).

The screenshot displays the 'Add AWS Deployment' form in the SMARTUNIFIER interface. The form includes the following fields and settings:

- Instance:** demoscenario.csv2rest:SUInstance:latest (Callout 4)
- Endpoint ID:** demo-AWSAccount1 (Callout 5)
- VPC:** vpc-0b7318756fa7fcfab
- Subnet:** subnet-09ec948d12803ed24
- Security Group:** SMARTUnifierIntegrationTestECSContainer_SG
- Role:** arn:aws:iam::[account-id]:role/SMARTUnifierIntegrationTestCodeBuildServiceRole
- S3bucket:** smartunifier-integration-test
- Cluster:** arn:aws:ecs:eu-central-1:[account-id]:cluster/SMARTUnifierIntegrationTestCluster
- Repository:** su-instance
- CPU:** .25 vCPU
- Memory:** 512
- Log File Configuration:** Info (Callout 6)
- Enable Encryption:** ☐ (Callout 7)
- Protected:** ☐ (Callout 8)
- Save Button:** (Callout 9)

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

Deployment						
Group ↑	Name	Version	Deployment Type	State		
su.demo.dashboard	SUInstance	1.0.0	AWS	NotDeployed		

- 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.

4.5 How to deploy, run and operate a deployed Instance

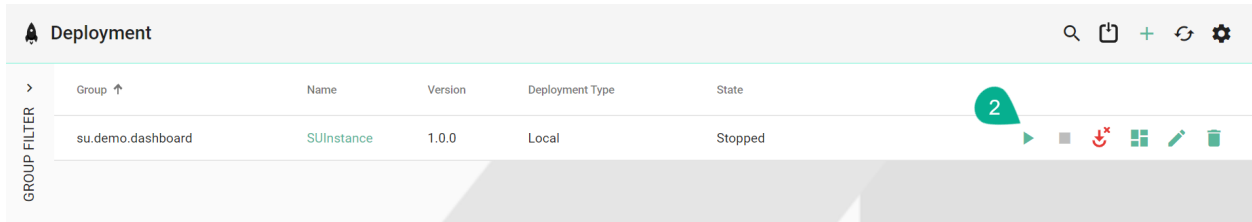
4.5.1 How to deploy an Instance

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

Deployment						
GROUP FILTER	Group ↑	Name	Version	Deployment Type	State	
	su.demo.dashboard	SUInstance	1.0.0	Local	NotDeployed	

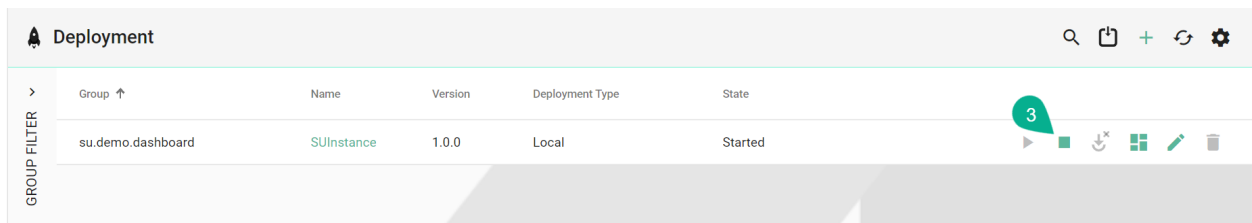
4.5.2 How to run an Instance

- 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.



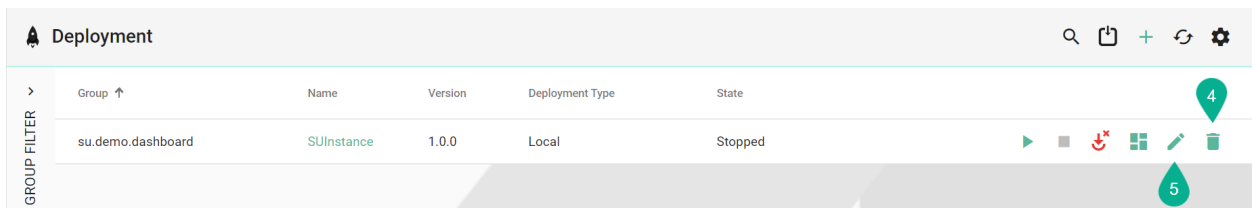
4.5.3 How to stop an Instance

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



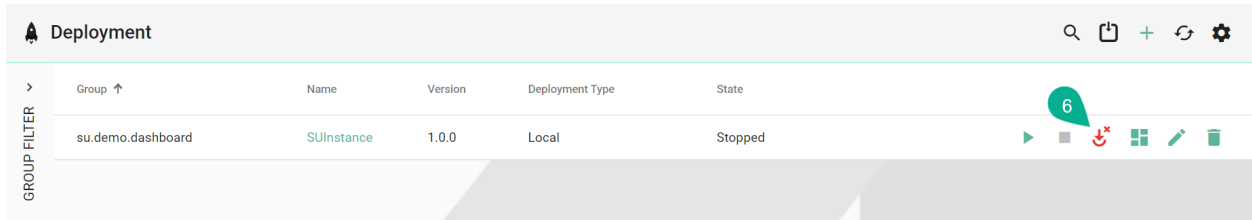
4.5.4 How to delete a Deployment of an Instance

- 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.



4.5.5 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.



4.6 How to monitor a deployed Instance

4.6.1 Log Viewer

SMARTUNIFIER comes with an integrated log viewer, which helps to gain insights in deployed and running Communication Instance.

The log viewer will show the details of logs based on the level defined throughout the creation of the deployment.

Log Levels

TRACE The most fine-grained information only used in rare cases where full visibility of what is happening inside a Communication Instance.

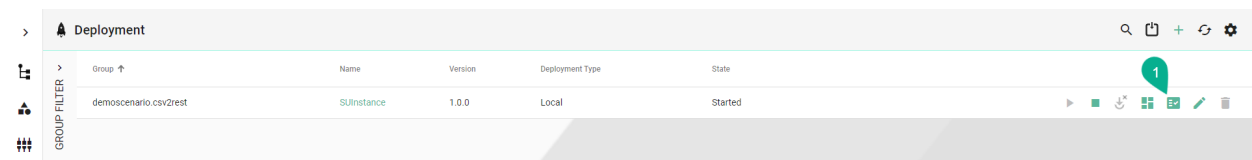
DEBUG Less granular compared to the TRACE level, but more than needed in an production environment. The DEBUG log level should be used for troubleshooting an faulty Communication Instance or when running a Communication Instance inside a test environment.

INFO Is the standard log level used for a standard deployment of a Communication Instance.

WARNING Log level that indicates that something unexpected happened inside a Communication Instance that might cause problems for the course of communication.

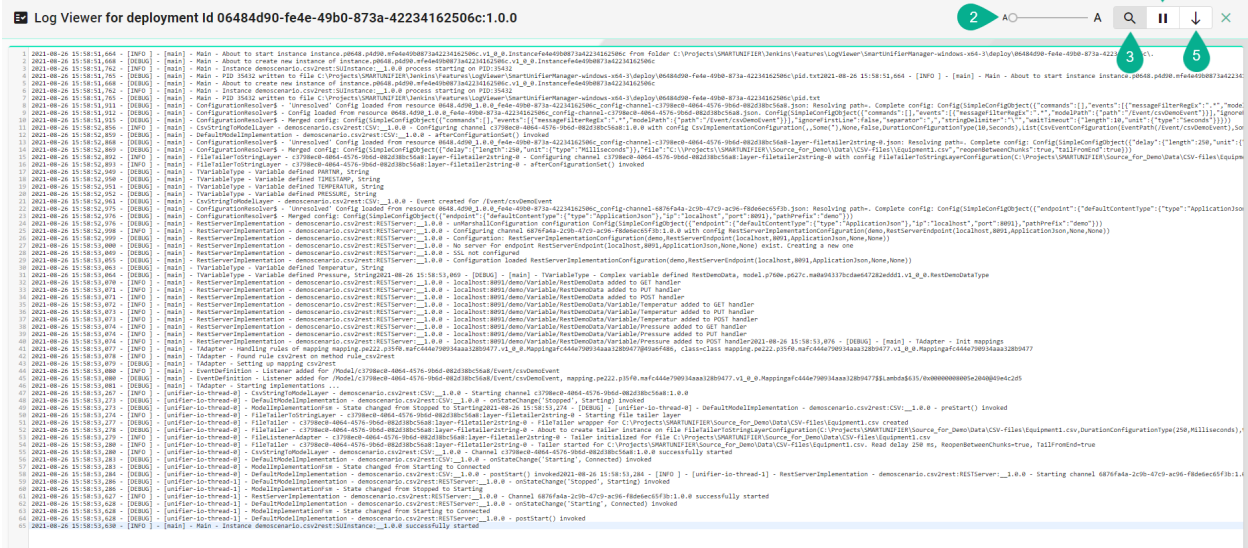
Log Viewer operation

Logs can be accessed by clicking the “Log” button (1).



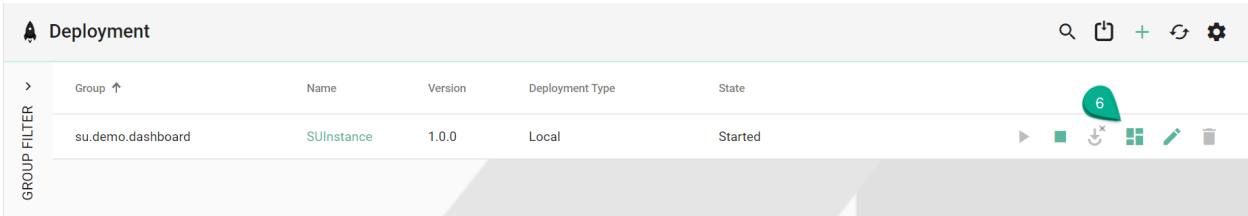
Log Viewer comes with the following features:

- Font size adjustability (2)
- Searching, based on a regular expression (Regex) (3)
- Start/Stop to “freeze” the current logging in order to investigate already printed log lines (4)
- Follow Tail, to skip through to the latest log line (5)



4.6.2 Dashboard

- In order to monitor an Instance, access the Dashboard view by clicking the “Dashboard” button (6).
- If the Instance is in the state NotDeployed the Dashboard cannot be accessed.



- 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

4.6. How to monitor a deployed Instance

overview su.demo.dashboard:SUIInstance:1.0.0

Channels

Info	Type	status	Model
su.demo.dashboard:SiemensS7PLC:1.0.0	Unknown	Connected	su.demo.dashboard:SiemensS7PLC:1.0.0
su.demo.dashboard:MESSimulator:1.0.0	Unknown	Disconnected	su.demo.dashboard:MESSimulator:1.0.0
su.demo.dashboard:PLCToInfluxDb:1.0.0	Unknown	RunningFailure	su.demo.dashboard:Analytics:1.0.0

Mappings

Info	models
su.demo.dashboard : PLCToInfluxDb : 1.0.0	db, plc
su.demo.dashboard : PLCToMESSimulator : 1.0.0	flow, plc

status Started

Tim... 2021-07-23 13:07:58

Time Up 0:01:27

cpuUsage (0%)

memoryUsage (1.77%)

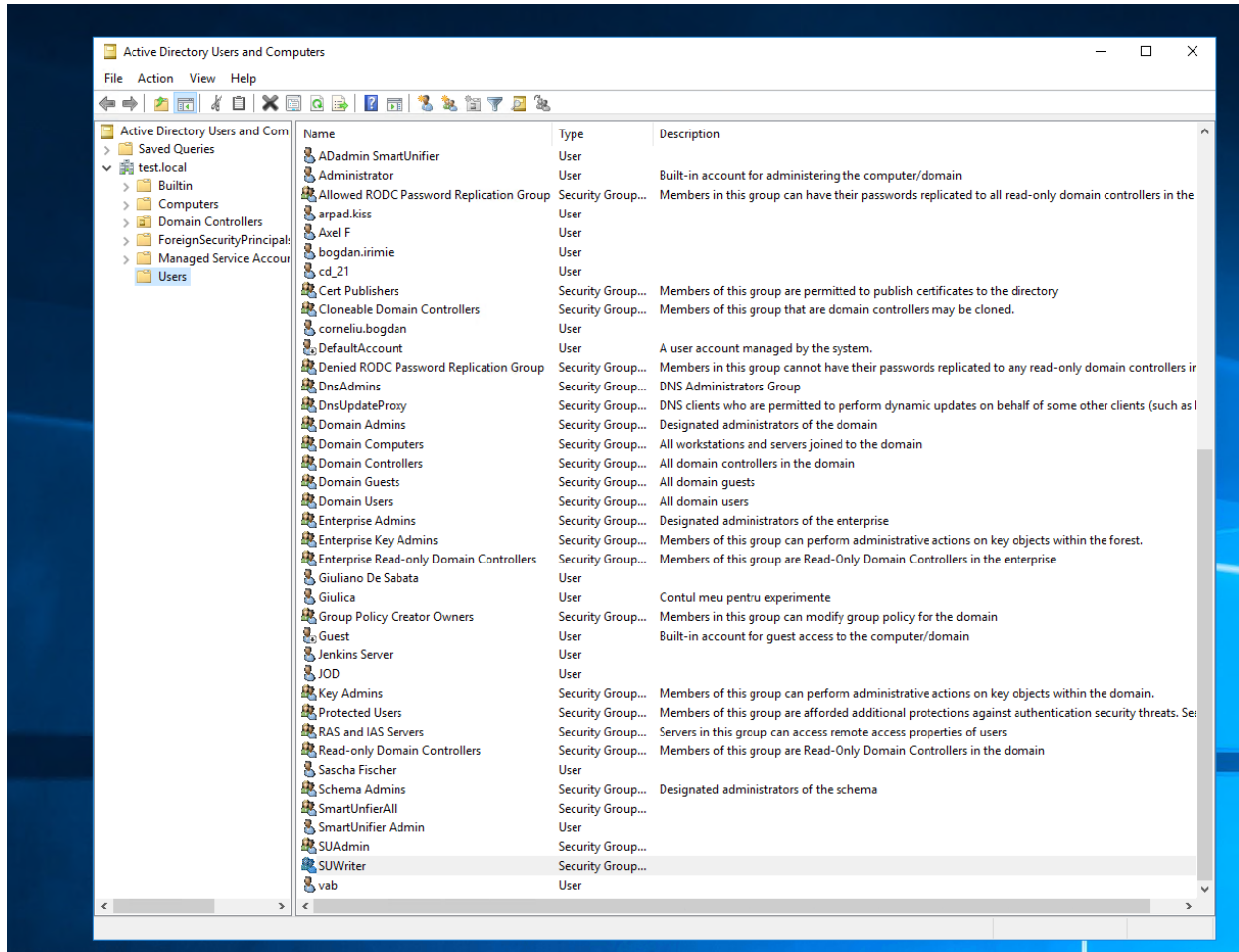
ADMINISTRATION

Learn how to:

- Integrate an *Active Directory*
- *Backup* and *Restore* the Repository
- Manage *Communication Channel Types*
- Manage *Docker Java Images*
- Create *Deployment Endpoints*
- Manage *User Accounts*

5.1 Active Directory Integration (ADI)

SMARTUNIFIER supports Windows Active Directory (AD). System administrators can use the Active Directory to add/remove users, groups, and resources quickly and efficiently through one dashboard.



5.1.1 AD Group Mapping

An user from AD must be added to a group that acts as a role. The role determines what permissions are assigned to the user.

The mapping between the AD groups and the SMARTUNIFIER roles is defined in the **application.conf** file from the **conf** folder.

```

1  # https://www.playframework.com/documentation/latest/Configuration
2  include "default.conf"
3  apiPrefix = adapter
4
5  play = {
6    server = {
7      http.port = 9000
8      http.address = "0.0.0.0"
9
10     #http.port=disabled
11     #https.port=9443
12     #https.keyStore.path="path_to_keystore"
13     #https.keyStore.password="keystore_password"
14   }
15   http.secret.key = "ChangeMySecret"
16 }
17 authentication {
18   activedirectory {
19     host = "192.168.0.132",
20     port = 389,
21     baseDN = "DC=test,DC=local",
22     useSSL = false,
23     user = "jenkins@test.local",
24     password = "Aiurea05",
25     groupmapping = {
26       Administrator = "SUAdmin",
27       Writer = "SUWriter",
28       Reader = "SmartUnifierAll"
29     }
30   }
31 }
32 unifiermanager {
33   tempFolder = "temp"
34   compiler {
35     scala {
36       javaHome = "jre"
37     }
38     management {
39       dontDeleteWorkspace = true
40     }
41   }
42   model {
43     loadCodeFromScalaFile = false
44   }
45   deployment {
46     javaHome = "jre"
47     local = {
48       deploymentFolder = "deploy"
49       softRefreshInterval = 2000
50       hardRefreshInterval = 5
51       logStatusInterval = 30
52       monitorLogs = true
53     }
54     docker {
55       baseimage {
56         autocreate = false
57         jreImage = "adoptopenjdk/11-jre-hotspot"
58       }
59     }
60   }
61 }

```

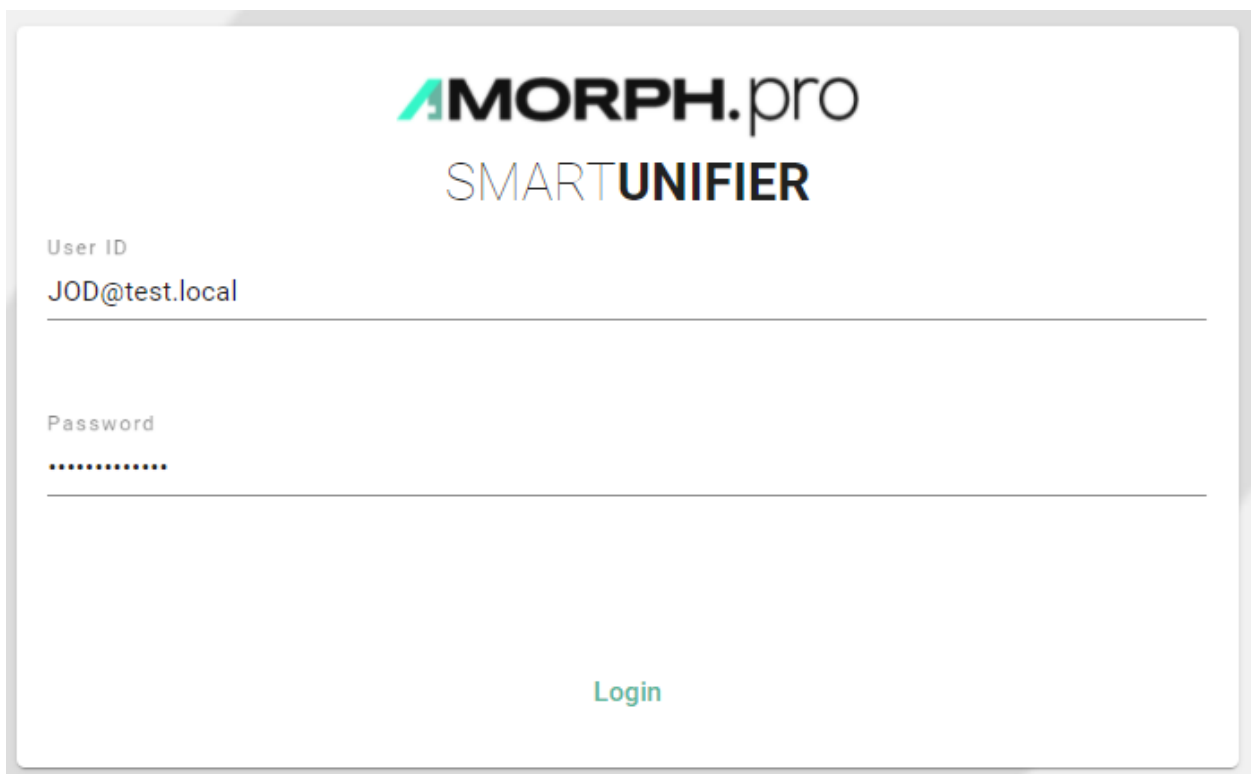
As seen above (1) in the left side are the SMARTUNIFIER roles and in the right side, between the quotation marks are the AD groups.

The SMARTUNIFIER roles are predefined:

- Administrator - global permission
- Writer - limited permission, write and read access
- Reader - limited permission, read access

A user from an AD group will have permission based on the mapping of the AD group to a predefined SMARTUNIFIER role.

After all the above configuration is done, the user can login to the SMARTUNIFIER with the **User logon name** and the **Password** defined in AD.

The image shows a login interface for SMARTUNIFIER. At the top, there is a logo for 'AMORPH.pro' in a stylized font, with 'AMORPH' in black and '.pro' in a lighter grey. Below the logo, the word 'SMARTUNIFIER' is displayed in a bold, black, sans-serif font. Underneath the title, there are two input fields. The first field is labeled 'User ID' in a small, grey font, and it contains the text 'JOD@test.local'. The second field is labeled 'Password' in a small, grey font, and it contains a series of dots representing a masked password. Below these fields, there is a 'Login' button with a green outline and the word 'Login' in a green font.

5.2 Backup and Restore

SMARTUNIFIER provides the possibility to manually backup and restore the repository. The repository represents a central location in which all the configuration components are stored:

- Information Models
- Communication Channels
- Mappings
- Node Types

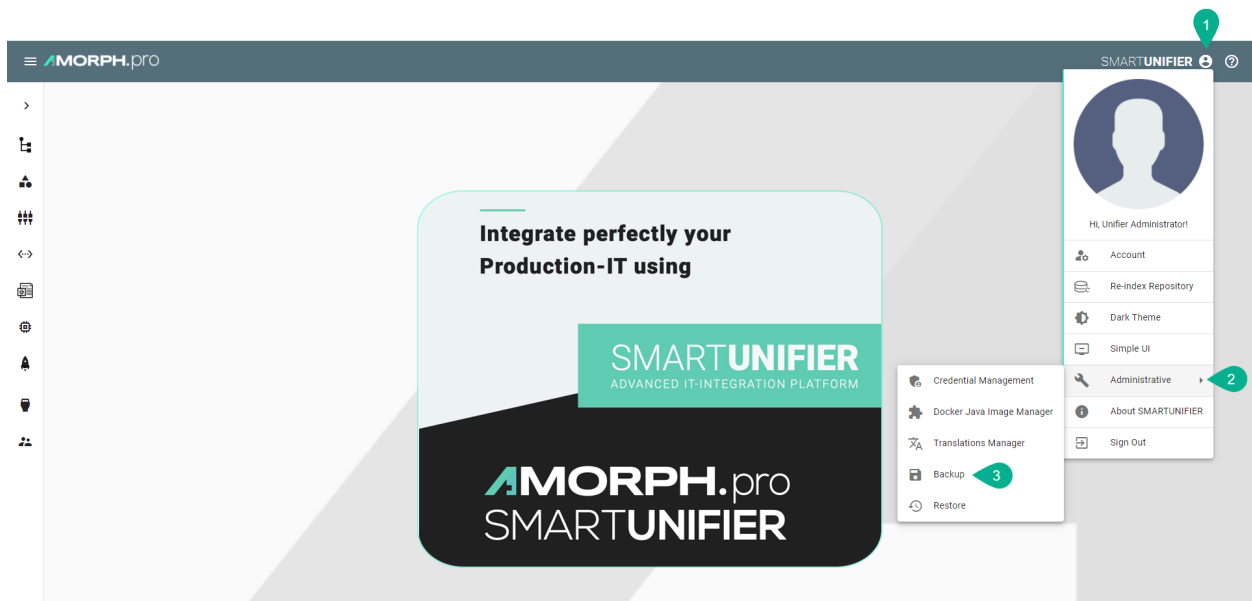
- Instances

5.2.1 Backup

The Backup feature provides the possibility to create a copy of all the configuration components to store elsewhere, so that it can be used to restore the last used after a data loss event occurs.

Follow the steps described below to create a backup of the repository:

- Select the **Account** icon (1), go to the **Administrative** section (2) and select the **Backup** option (3).



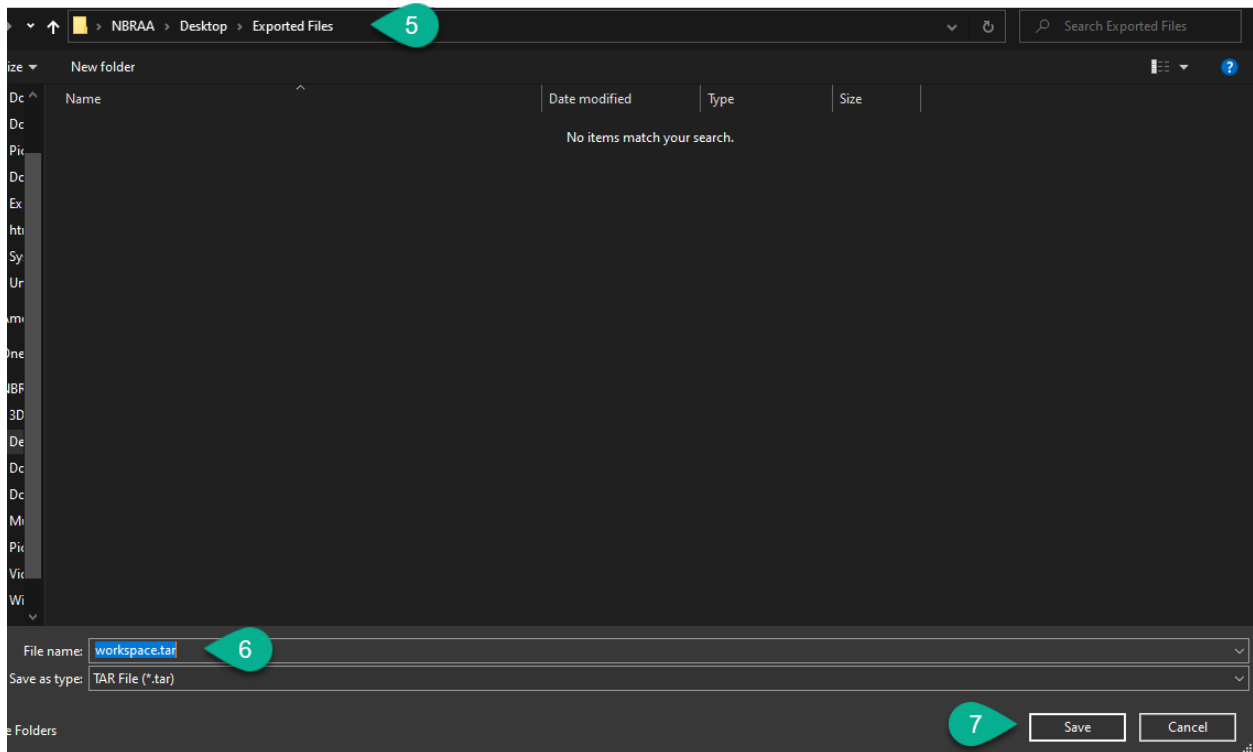
- Select **Yes** on the confirmation dialog (4) to continue

Backup Repository?

Are you sure you want to backup the Unifier Repository? This might take a long time based on the size of the repository



- Choose the path (5) and the name (6) to save the repository **TAR** file then click on the **Save** button (7) to finish.



- The **TAR** file contains all the SMARTUNIFIER configuration components:
- Information Models
- Communication Channels
- Mappings
- Device Types
- Communication Instances

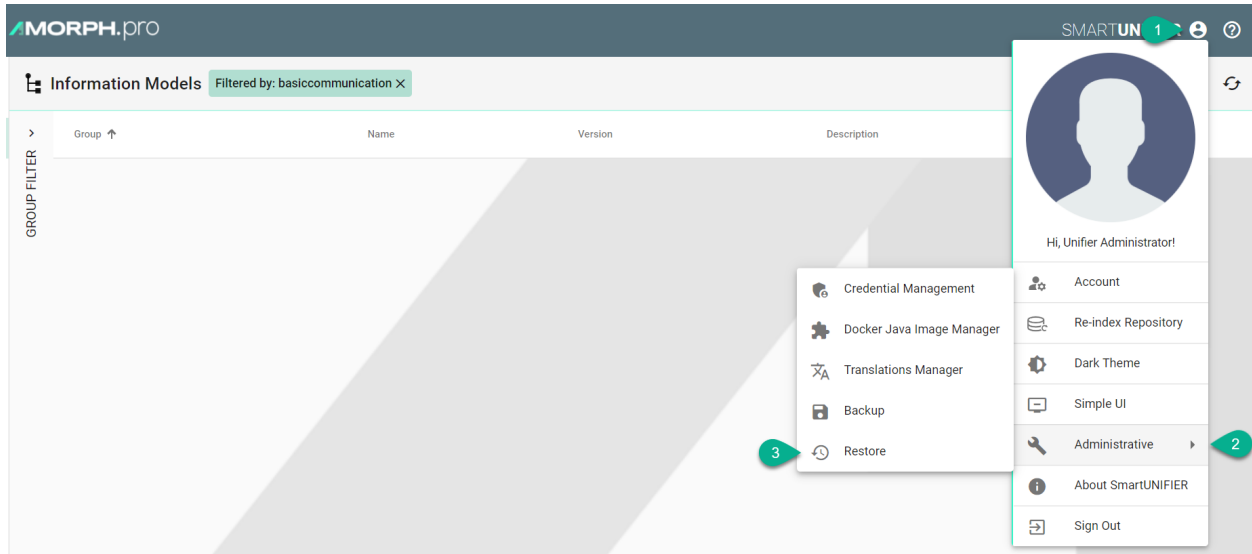
5.2.2 Restore

The Restore feature allows to copy the SMARTUNIFIER configuration components from a backup to the original location.

Note: When restoring, the existing configuration components will be overwritten by with the selected configuration components from the backup!

Follow the steps described below to restore the SMARTUNIFIER repository:

- Select the **Account** icon (1) , go to the **Administrative** section (2) and select the **Restore** option (3).



- A pop-up appears, choose the **TAR** file to restore (4) and select the **Yes** button (5) to finish.

Restore Repository?

Are you sure you want to restore a Unifier Repository? All existing data will be overwritten!



- The configuration components are uploading and all existing data will be overwritten!

5.2.3 Manager Backup

In order to backup SMARTUNIFIER Manager make a copy of the SMARTUNIFIER installation package.

Before the backup make sure to remove the following directories:

- temp
- workspace
- log
- deploy

Name	Date modified	Type	Size
bin	31.08.2021 07:10	File folder	
conf	07.09.2021 17:16	File folder	
jre	31.08.2021 07:10	File folder	
keystore	07.09.2021 17:16	File folder	
lib	31.08.2021 07:10	File folder	
licenses	31.08.2021 07:10	File folder	
log	07.09.2021 17:16	File folder	
manual	31.08.2021 07:10	File folder	
repository	31.08.2021 07:10	File folder	
scala	31.08.2021 07:10	File folder	
versioning	07.09.2021 17:16	File folder	
UnifierManager.bat	31.08.2021 07:10	Windows Batch File	1 KB

5.3 Channel Types Manager

By default, the Channel Types Manager displays all *Channels* included in your current version of SMARTUNIFIER.

Communication Channels that should be used within the configuration of a SMARTUNIFIER Communication Instance have to exist in the Channel Types Manager. How to add new Channel Types is shown in the *section below*.

Note: The Channel Types Manager can only be accessed by user accounts with an administrator role assigned.

5.3.1 About Layers

Implementations of SMARTUNIFIER Communication Channels consist of one and up to three so-called layers.

The target of layers is to transform data from Information Models into the respective data format of the specific protocol used in case the data traffic is outgoing from a SMARTUNIFIER Communication Instance. The same principle applies when data is incoming.

As an example for such a layer stack you can see below the layer stack for the *MQTT Communication Channel*:

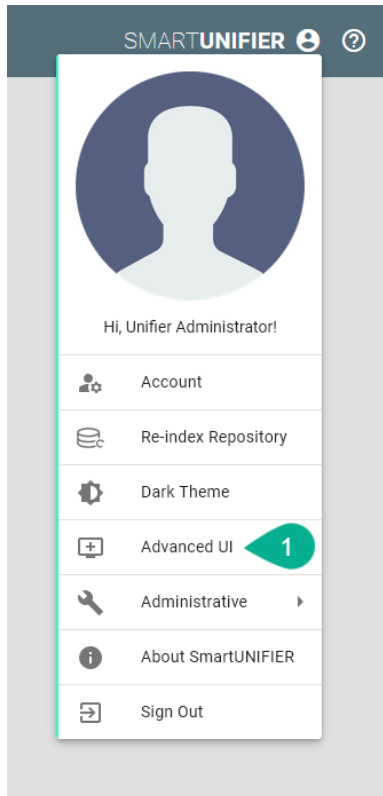
- Data that is incoming from a Device is transformed into a String behind the scene.
- The String is then converted into a JSON Object.

- Finally, the JSON is used to assign data to the assigned Information Model.

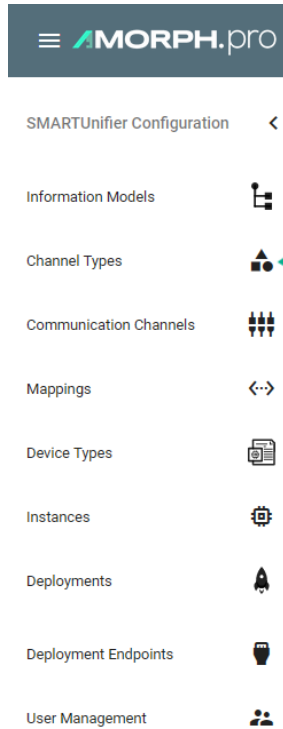
5.3.2 How to create a new Channel Type

Follow the steps below to create a new Channel Type:

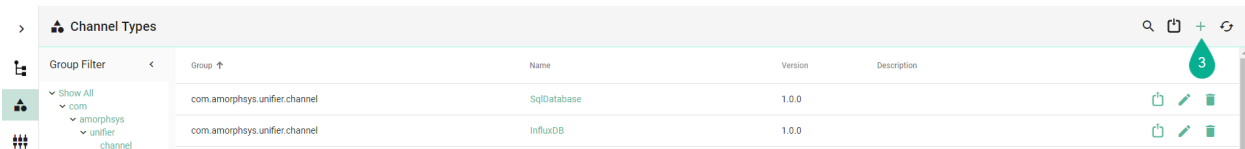
1. Open the SMARTUNIFIER menu and select **Advanced UI**.



1. Go to the Channel Types perspective by clicking the **Channel Types** button.



1. Click on the **Add** button in the upper right corner.



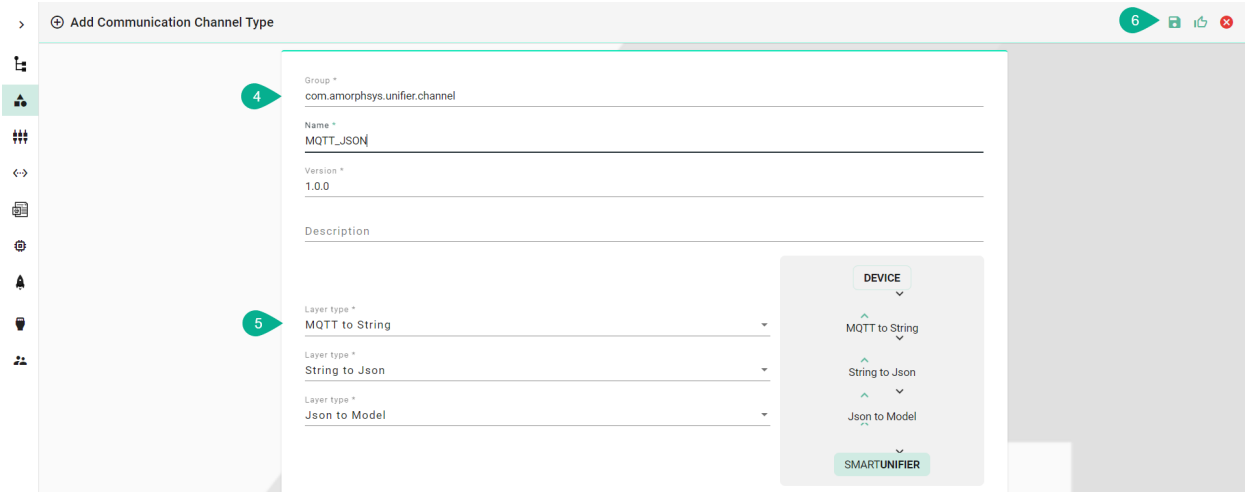
4. Enter some descriptive information:

- Enter a **group**
- Enter the **name** of the Channel
- Enter a **version**

5. Next, define the layer stack of the new Channel Type:

- Select a layer with the **Layer type** drop-down menu.
- In case the selected layer has more layers dependent on itself, select again another layer with the **Layer type** drop-down menu showing up below.

6. To save the Communication Channel Type select the **Save** button.



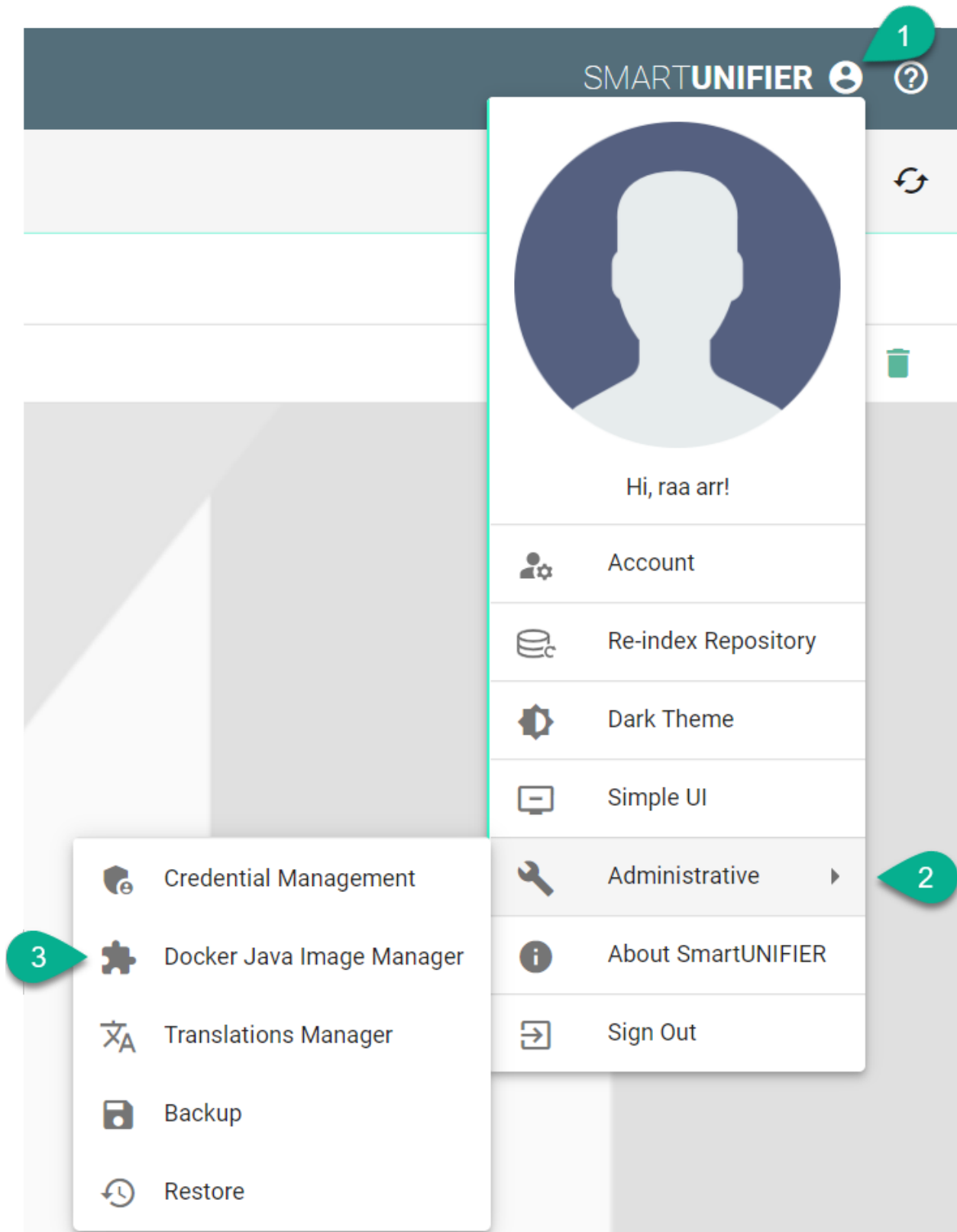
5.4 Docker Java Image Manager

SMARTUNIFIER supports the Deployment of Instances using Docker Containers using different Java base images. With the Docker Java Images Manager, the user can create and maintain different versions of Docker Java images.

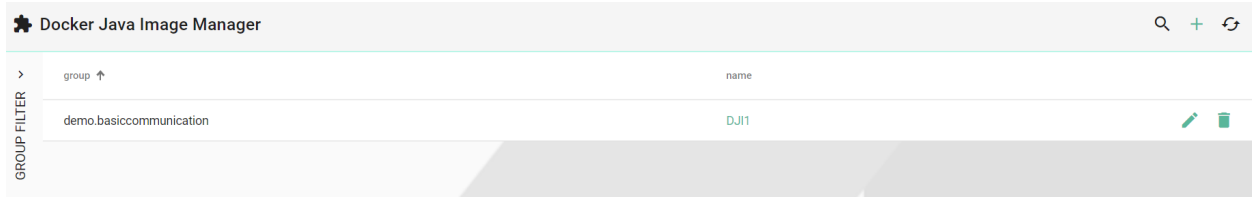
This feature can only be accessed by a user with the administrator role.

5.4.1 Access to the Docker Java Image Manager

To open the Docker Java Image Manager, select the **Account** icon (1), go to **Administrative** section (2) and select the **Docker Java Image Manager** option (3).



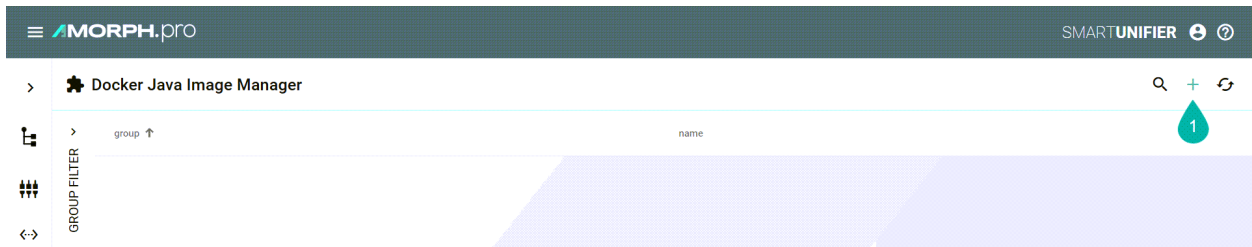
The Docker Java Image Manager is visible.



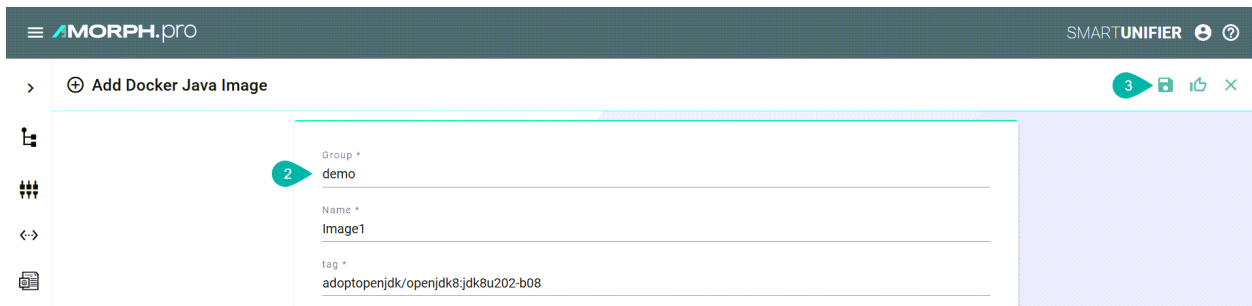
5.4.2 Add a New Docker Java Image

Follow the steps described below to add a new Docker Java image:

- Click on the **Add** button (1).

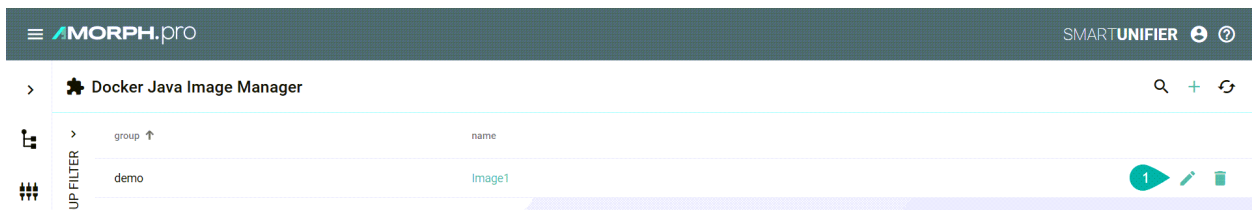


- In the *Add Docker Java Image* view, a set of configuration parameters is required (2): * Provide a **Group** and a **Name** * Provide a **tag** e.g., adoptopenjdk/openjdk8:jdk8u202-b08
- After all mandatory fields are filled in, click the **Save** button (3).



5.4.3 Edit a Docker Java Image

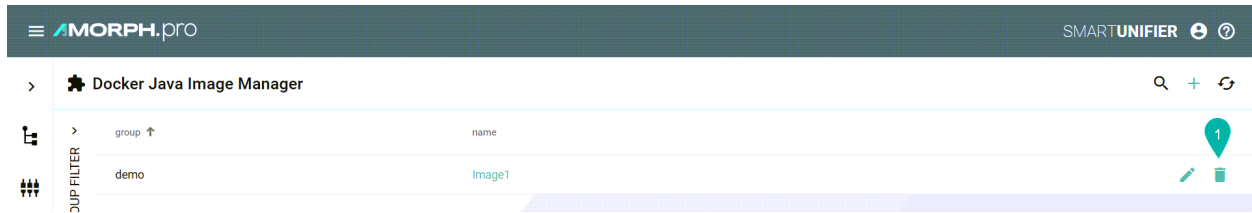
To edit a Docker Java image, select the **Edit** button (1).



The Docker Java image is in the Edit Mode, the configuration parameters can be edited and then save the session by selecting the **Save** button.

5.4.4 Delete a Docker Java Image

To delete a Docker Java image, select the **Delete** button (1).



A pop-up confirmation appears, select the **Delete** button.

5.5 Deployment Endpoints

5.5.1 What are Deployment Endpoints

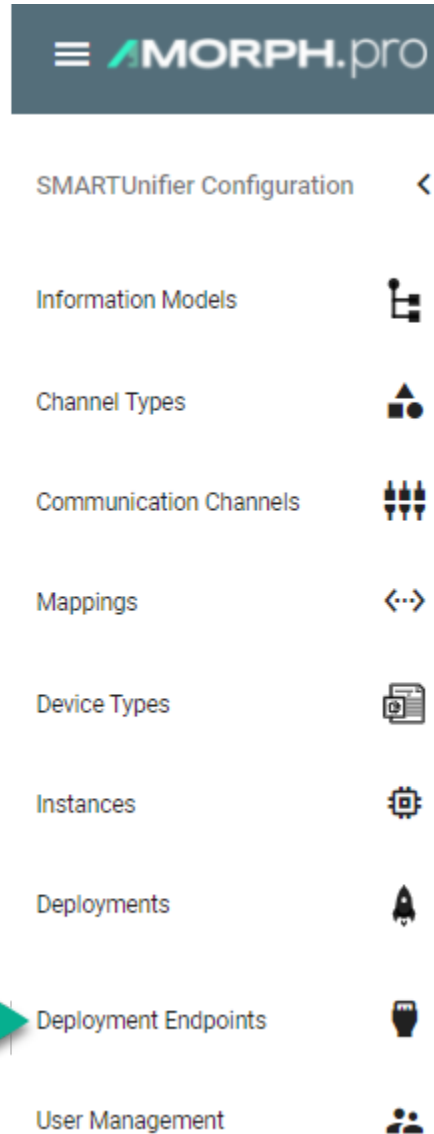
Deployment Endpoints are used to identify the location of a Deployment (i.e., the definition where an Instance is executed). With the Deployment Endpoints, you can create and maintain those locations. This feature can only be accessed by a user with the administrator role.

5.5.2 Deployment Endpoints Types

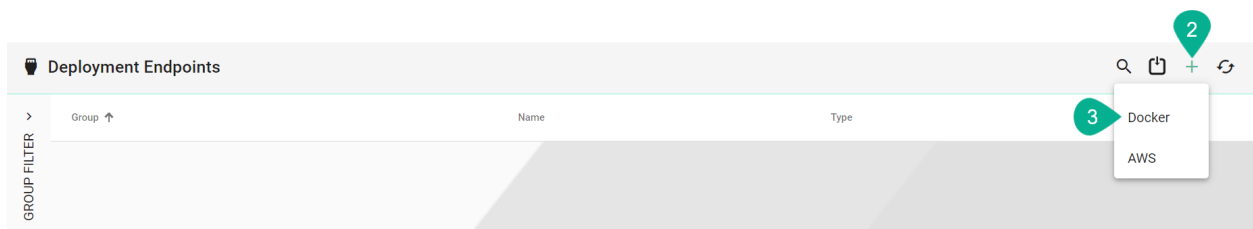
Docker

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 must 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**.

- If needed, enable **TLS** by enabling the checkbox
- After all mandatory fields are filled in, click the “Save” button (5).

⊕ Add Docker Endpoint

4

Group *

demo

Name *

DockerEndpoint1

URL *

tcp://127.0.0.1:2375

☐ Enable TLS

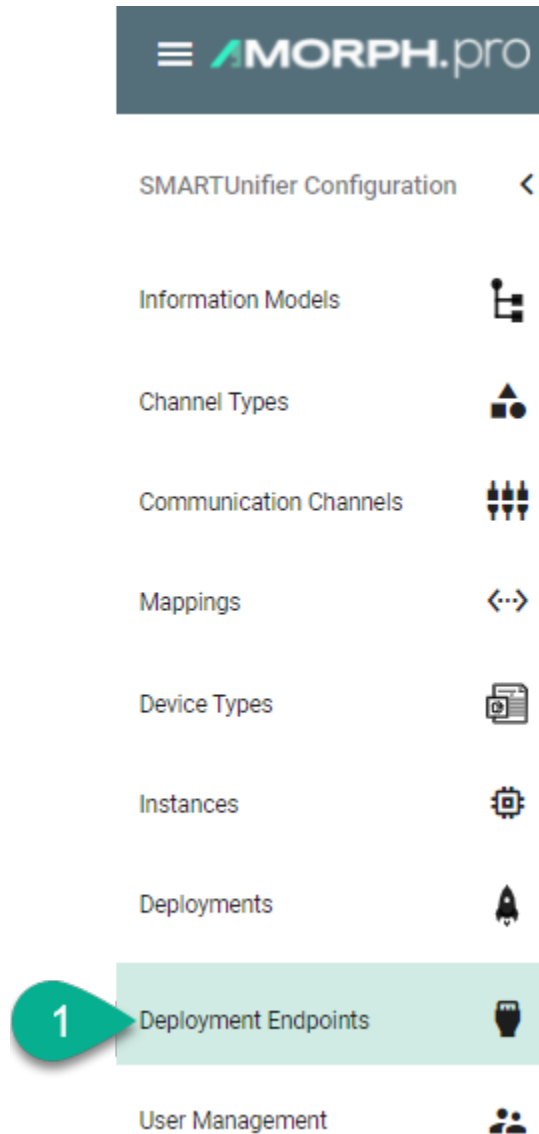
5

AWS

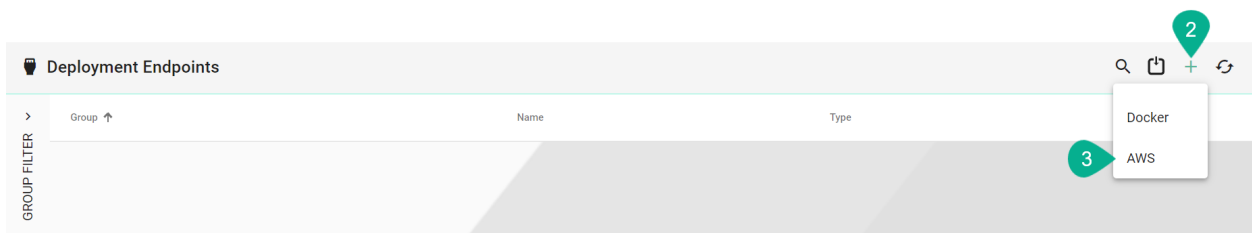
Before deploying a **SMARTUNIFIER** Instance on AWS Fargate you need to create an AWS Deployment Endpoint. The AWS Deployment Endpoint specifies, which AWS account should be used for the deployment.

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

5

⊕ Add AWS Endpoint

Group *

demo

Name *

AWSAccount1

Account ID *

Region *

eu-central-1

5.6 User Management

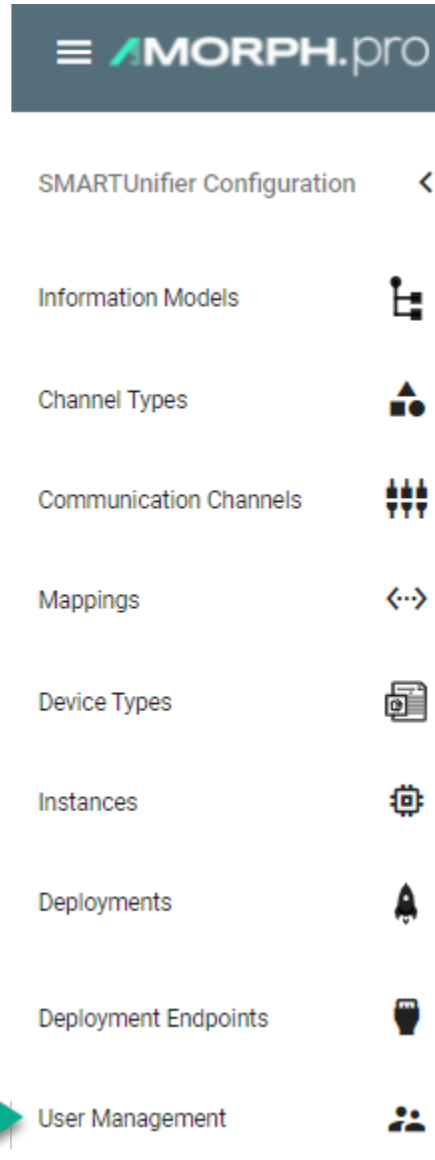
5.6.1 About User Management

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

5.6.2 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 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 role for the user. 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).

⊕ Add User

3

User ID *

JohnDoe2

Email

First Name *

John

Last Name *

Doe

Language *

English

Role: Writer

Administrator ☐

Reader ☐

Writer ☒

Status: Active

Active ☒

Inactive ☐

Credentials

Password *

9 / 10

- ✓ 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

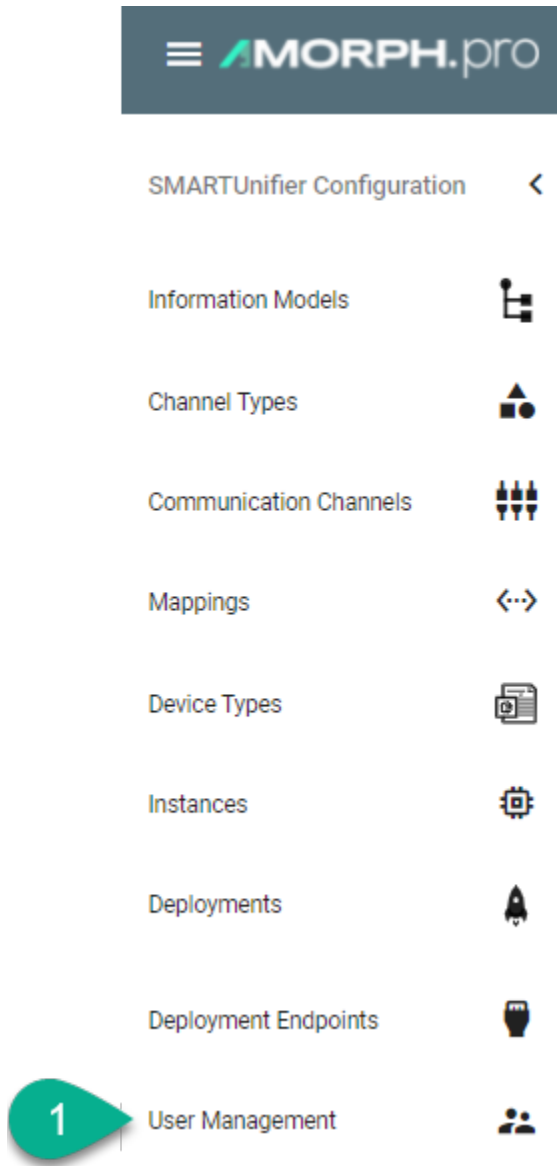
Confirm password *

4

5.6.3 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).

A screenshot of the 'User Management' table. The table has columns for User ID, Email, First Name, Last Name, Language, Role, Status, and Created. There are two rows of data. The first row is for 'JohnDoe2' and the second for 'admin'. To the right of each row, there are two icons: a pencil (edit) and a trash can (delete). A green callout bubble with the number '2' points to the pencil icon for the 'JohnDoe2' user.

User ID	Email	First Name	Last Name	Language	Role	Status	Created
JohnDoe2		John	Doe	en	Reader	Active	2021-03-26 00:00:00.000
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 ☐

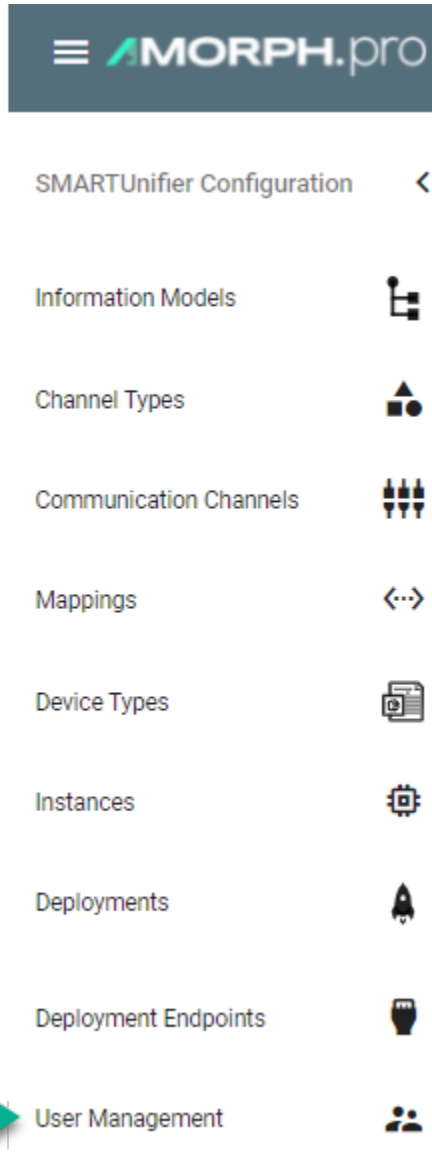
Change Password

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

5.6.4 Delete a user

This procedure describes how to delete a user account.

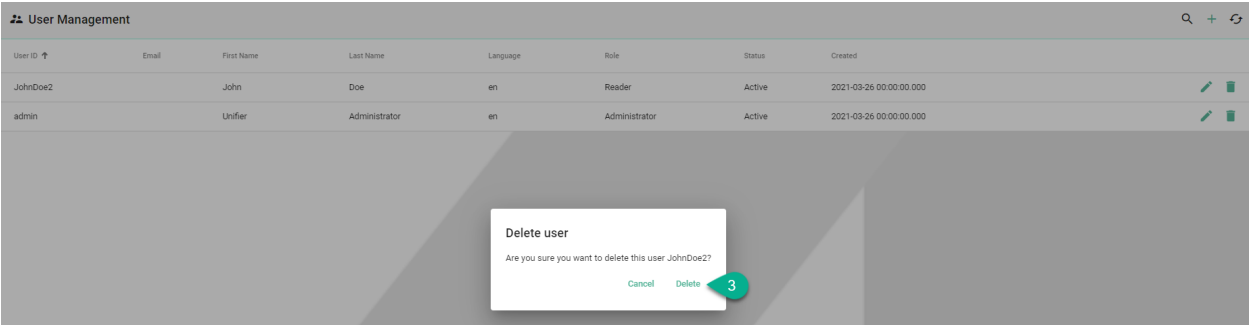
- Select the SMARTUNIFIER User Management perspective (1).



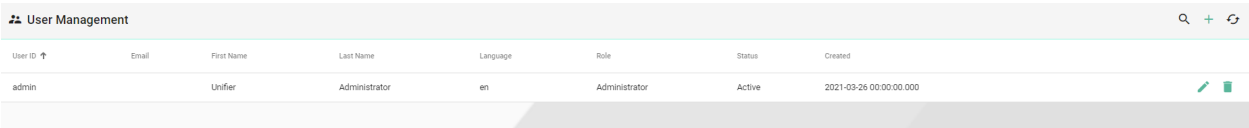
- Click the “Delete” button (2).

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.



DEMO SCENARIOS

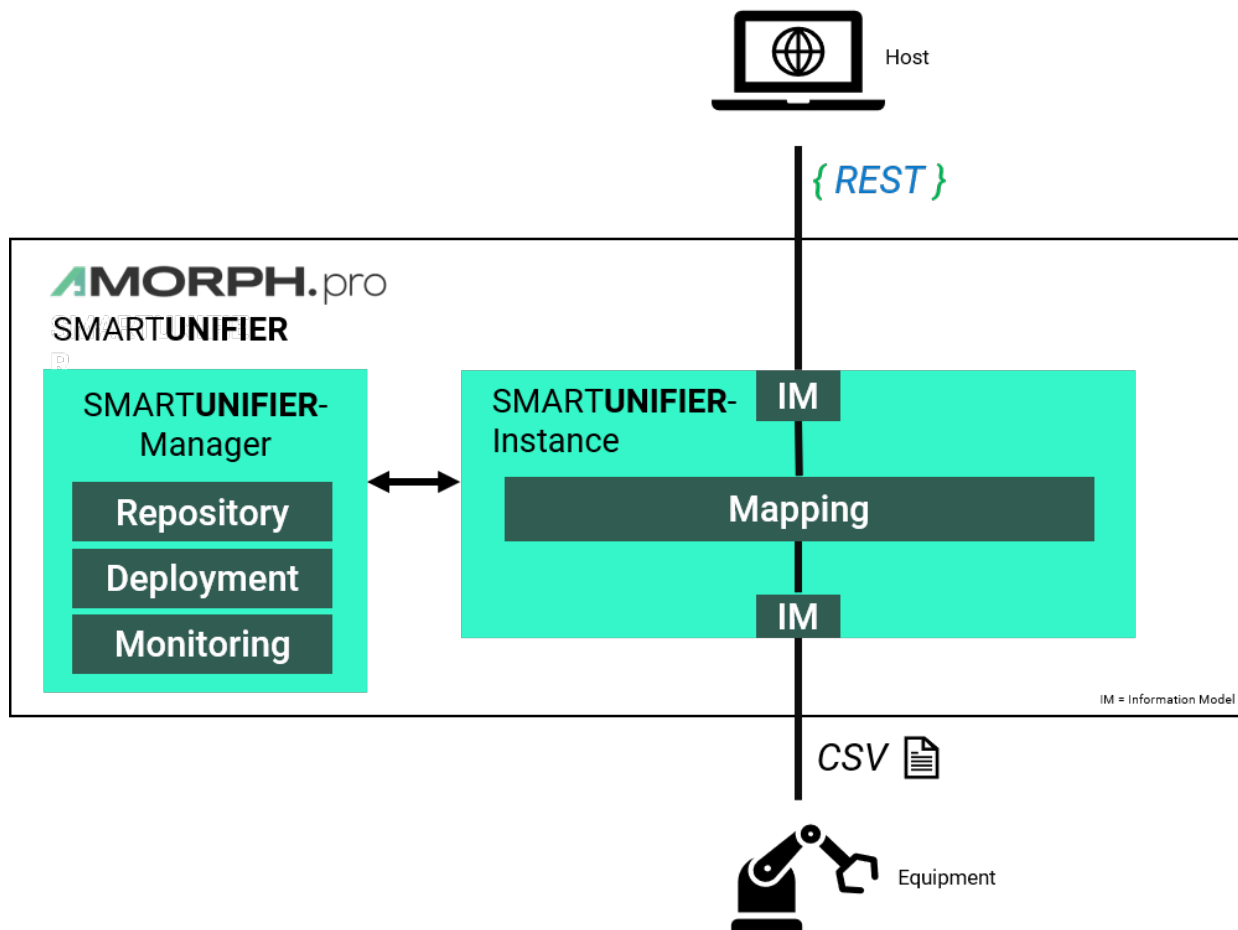
You want to learn and get your hands on a demonstration integration scenario? Check out the following demonstration uses cases:

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

6.1 File-based data - CSV to REST-Server

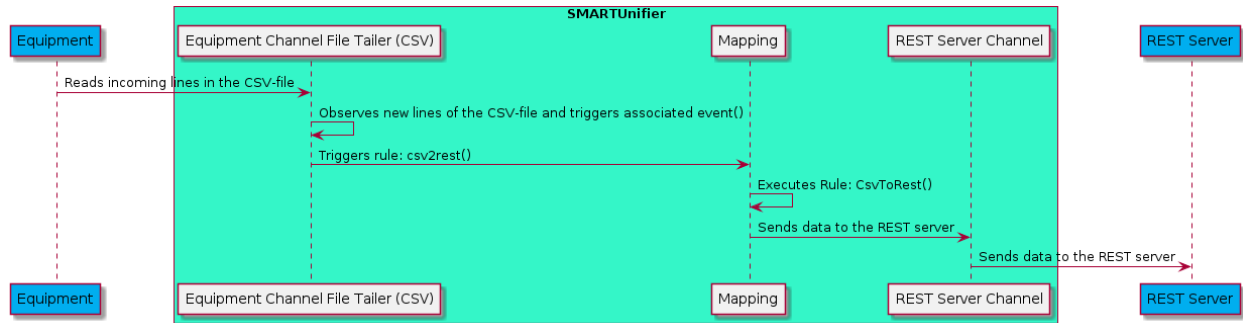
6.1.1 Overview

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"
```



6.1.2 Information Model

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 them 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 using the same “Group” name throughout the scenario. For example, to identify the created Artifacts for this scenario the group name “demoscenario.csv2rest” is used in this documentation.

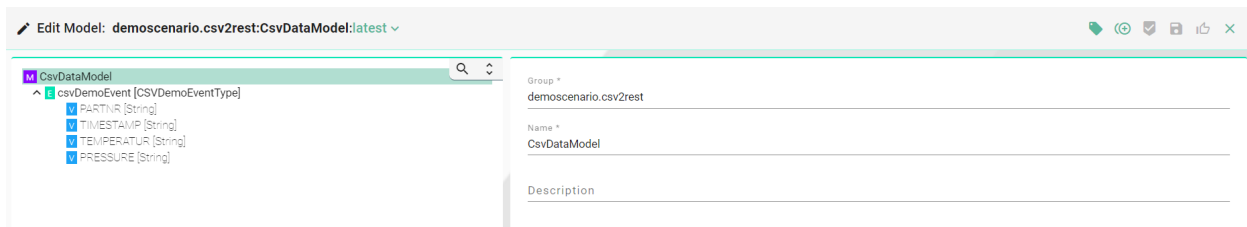


Table 1: CsvDataModel - Variables

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

Information Model - REST-Server

There must 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.

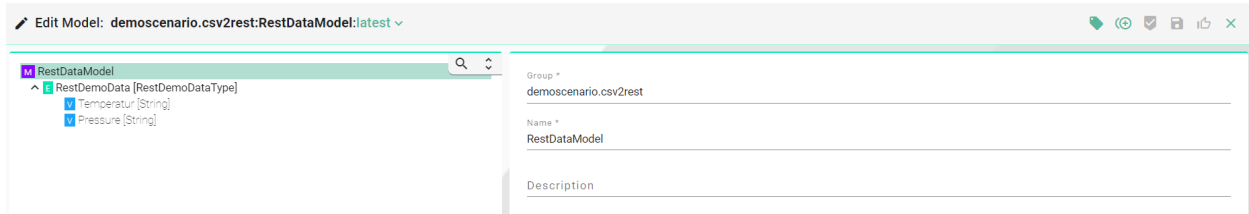


Table 2: RestDataModel - Variables

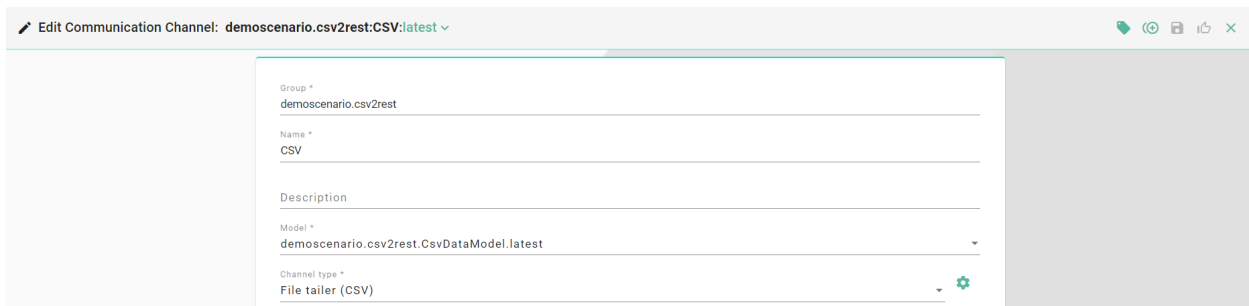
ID	Definition Type	Data Type
RestDemoData	Variable	RestDemoDataType
RestDemoData/Temperatur	Variable	String
RestDemoData/Pressure	Variable	String

6.1.3 Communication Channel

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.



Communication Channel - 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 “**demoscenario.csv2rest:RestDataModel:latest**”.
- 3.) Lastly, select as a Channel Type the “**RestServer**” Channel.

The screenshot shows the 'Edit Communication Channel' dialog for 'demoscenario.csv2rest:RESTServer:latest'. The fields are filled as follows:

- Group: demoscenario.csv2rest
- Name: RESTServer
- Description: (empty)
- Model: demoscenario.csv2rest.RestDataModel.latest
- Channel type: Rest Server

The configuration of the CSV-Channel, as well as the REST-Channel, is done in the section [Create Instance](#).

6.1.4 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 “**demoscenario.csv2rest:CsvDataModel:latest**”. Enter a Name for the Information Model, e.g., “**CsvDataModel**”.
- 3.) Click again the “Add” button (1) and select the Information Model “**demoscenario.csv2rest:RestDataModel:latest**”. Enter a Name, e.g., “**RestDataModel**”.
- 4.) Next, add a Rule to the Mapping. Click the “Add Rule” button (2).

The screenshot shows the 'Edit Mapping' dialog for 'demoscenario.csv2rest:CSVtoREST:latest'. It is divided into two panes: 'Configuration' and 'Rules'.

Configuration Pane:

- Group: demoscenario.csv2rest
- Name: CSVtoREST
- Description: (empty)
- Models list:

Short name	Information model identifier
CsvDataModel	demoscenario.csv2rest:CsvDataModel:latest
RESTDataModel	demoscenario.csv2rest:RestDataModel:latest

Rules Pane:

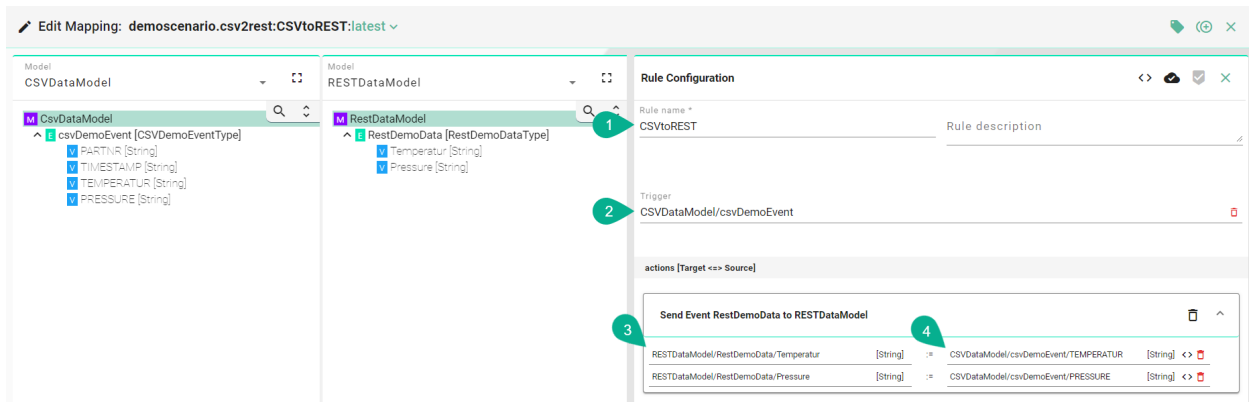
- Contains an 'Add Rule' button (2) and a search icon.

- 5.) Enter a Name for the new Rule (1) like “**csv2rest**”.

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)

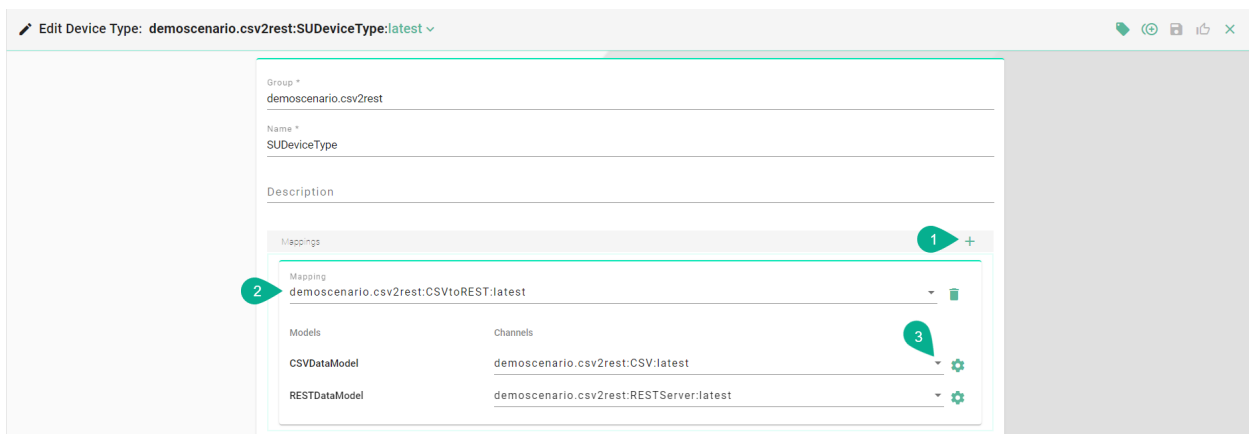
8.) Drag and drop the belonging variables “**TEMPERATURE**” and “**PRESSURE**” from the “**CsvDataModel**” into the Source fields (4)



6.1.5 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 “**demoscenario.csv2rest:CSVtoREST:latest**” previously created (2).
- 4.) Assign the correct Channels to the Information Models (3).
- 5.) Save the new Device Type.



6.1.6 Instance

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

- 1.) Select the Device Type “**demoscenario.csv2rest:SUDeviceType:latest**” previously created.
- 2.) Values for Group, Name and Version are already set from the Device Type. Although, we recommend changing the Name.

Edit Instance: demoscenario.csv2rest:SUInstance:latest

Device Type
demoscenario.csv2rest:SUDeviceType:latest

Group *
demoscenario.csv2rest

Name *
SUInstance

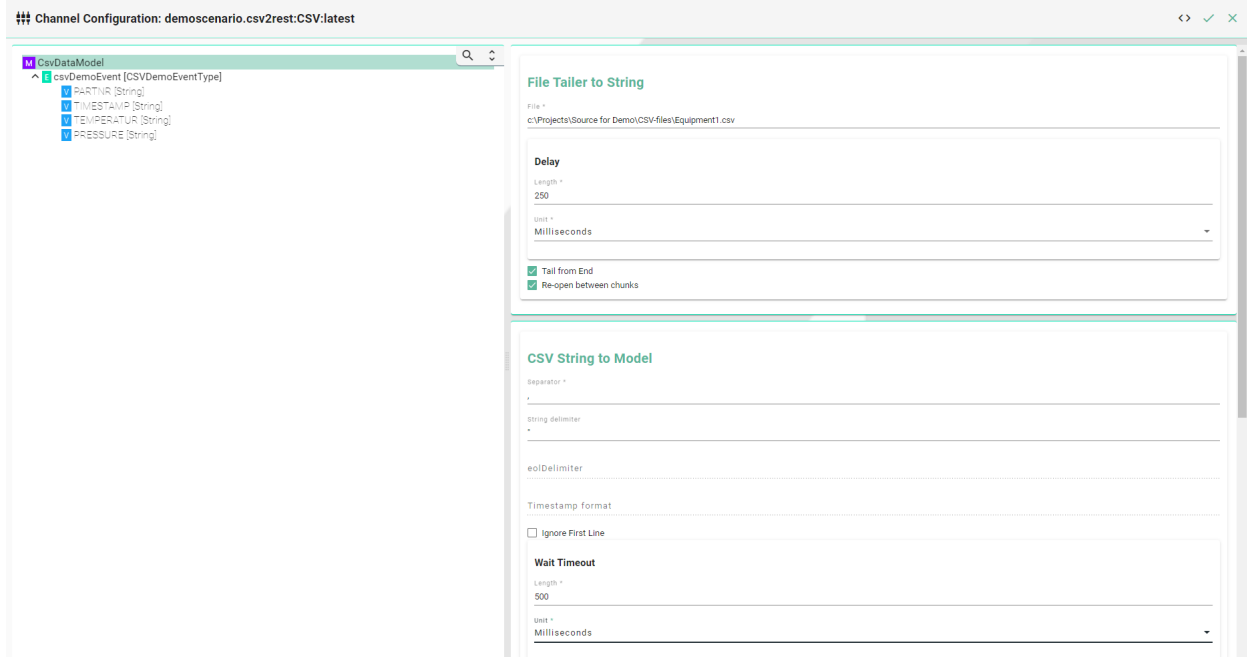
Description

Mappings

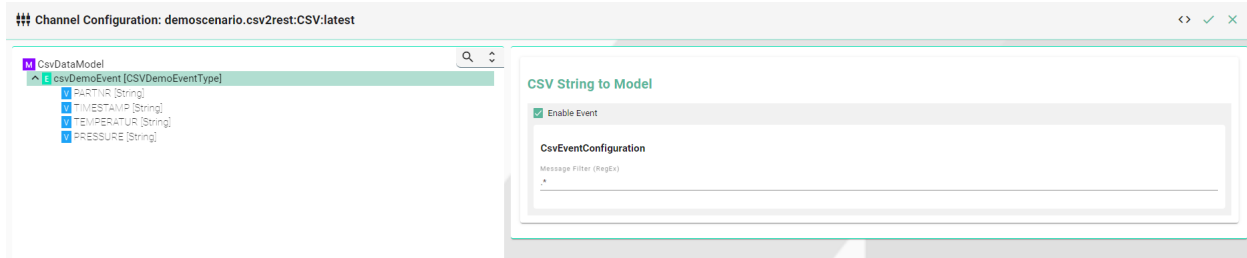
Mapping	Models	Channels
demoscenario.csv2rest:CSVtoREST:latest	CSVDataModel	demoscenario.csv2rest:CSV:latest
	RESTDataModel	demoscenario.csv2rest:RESTServer:latest

- 3.) Click the configuration button for the “**demoscenario.csv2rest:CSVChannel:latest**” 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.



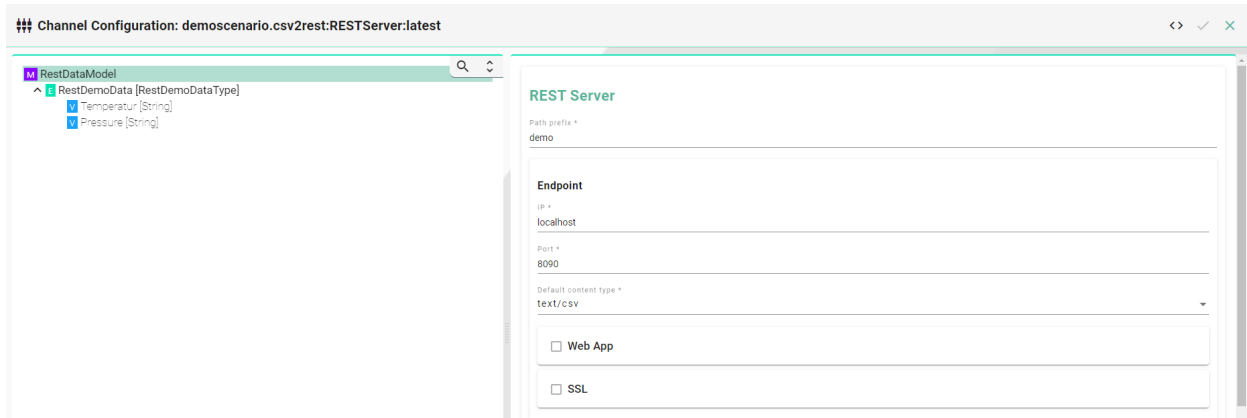
3.8) Select the “**csvDemoEvent**” node in the Information Model on the left side. Enter “.*” as the message filter RegEx.



4.) Lastly, it follows the configuration of the Rest Server Channel (3):

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

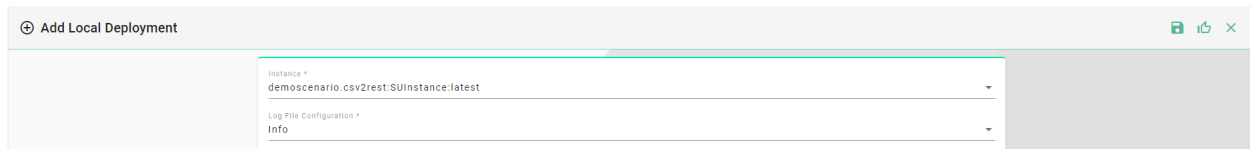
4.2) Leave the default settings for the Rest Server Endpoint as it is. We recommend changing the “**Port**” if the default is occupied.



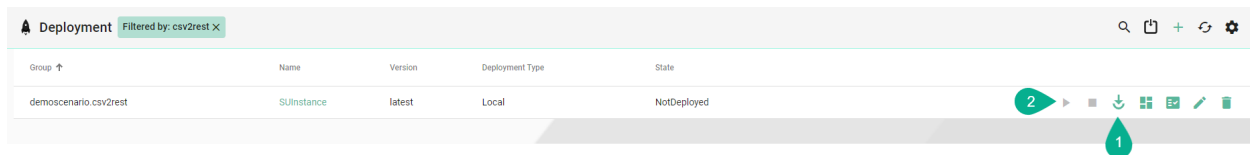
After all steps have been executed the Instance “**demoscenario.csv2rest:SUIInstance:latest**” is fully configured and ready for Deployment.

6.1.7 Deployment

- 1.) Select the Instance “**demoscenario.csv2rest:SUIInstance:latest**” 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.

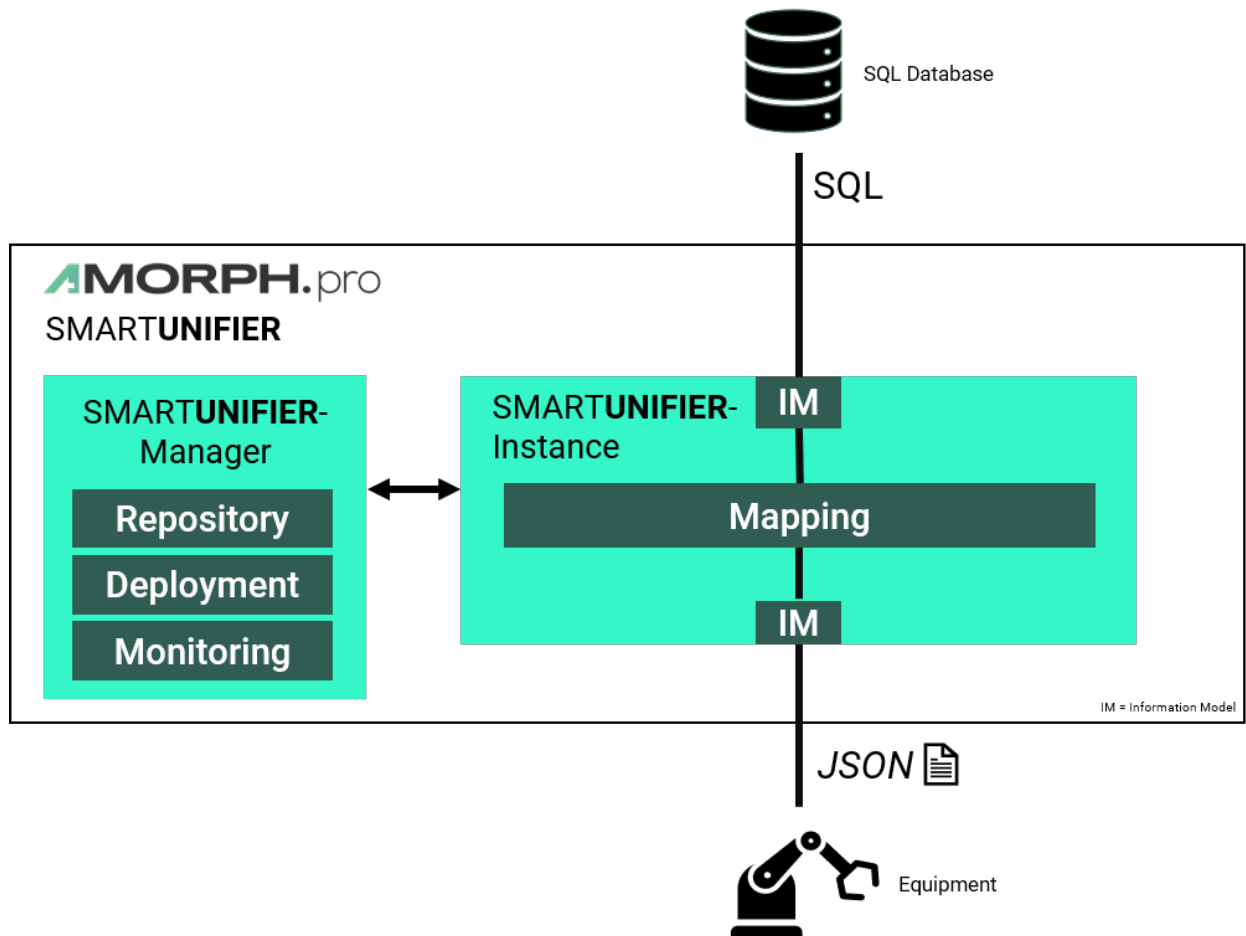
- Open any Browser and enter <http://localhost:8091/demo/Variable/RestDemoData> as an URL to receive the latest values for both Pressure as well Temperature.
- Open any Browser and enter <http://localhost:8091/demo/Variable/RestDemoData/Variable/Temperatur> as an URL to get the latest value for the value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

6.2 File-based data - Insert JSON data in SQL-Database

6.2.1 Overview

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



6.2.2 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.

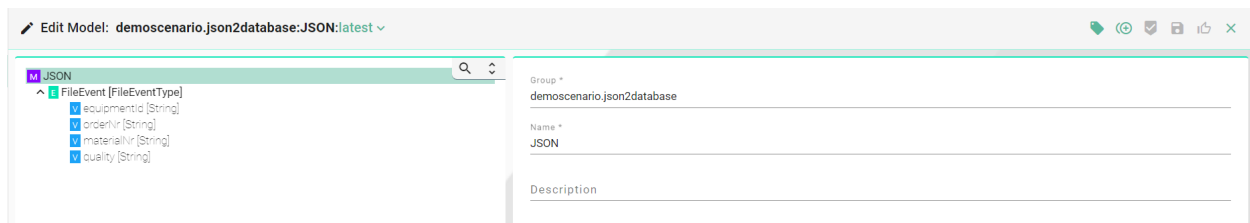
6.2.3 Information Model

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

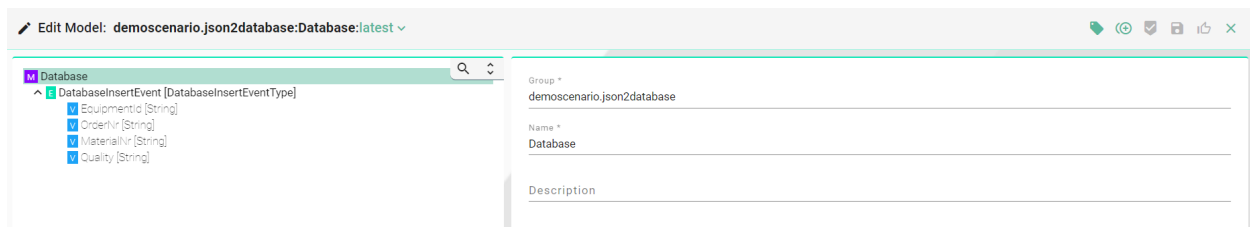


Information Model - SQL-Database

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



6.2.4 Communication Channel

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*

Communication Channel - SQL Database

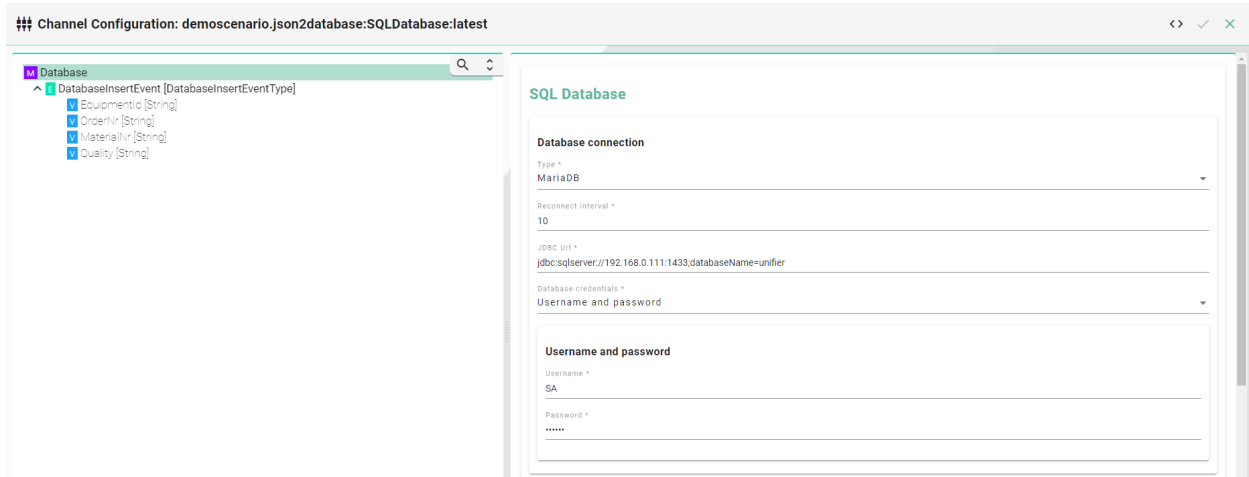
In this scenario the JSON-file is processed by the SMARTUNIFIER with the build-in File Consumer.

Create Database Channel:

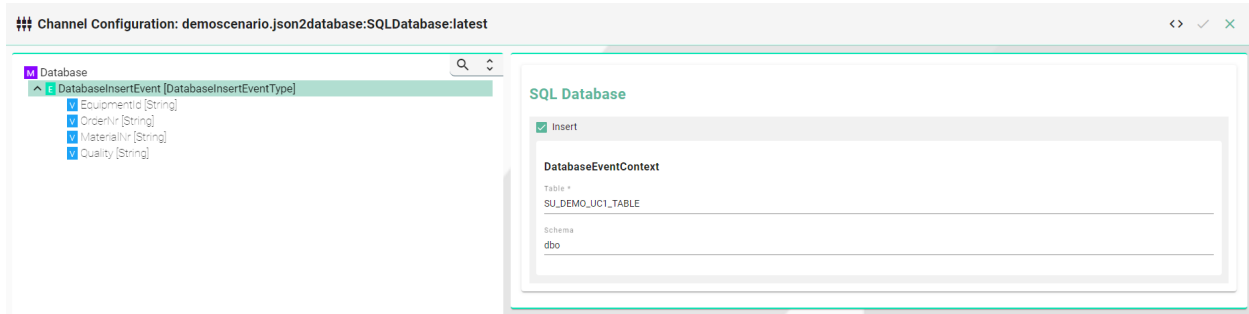
- Select the *Database* Information Model created previously
- Select *SqlDatabase* as Channel Type

Configuration:

- Select the root node of the Information Model and configure the database access
 - Select *SQLServer* as Type
 - Enter the **JDBC Url**, *according to the selected database type* - `jdbc:sqlserver://192.168.0.111:1433;databaseName=unifier`
 - Enter username and password of the database

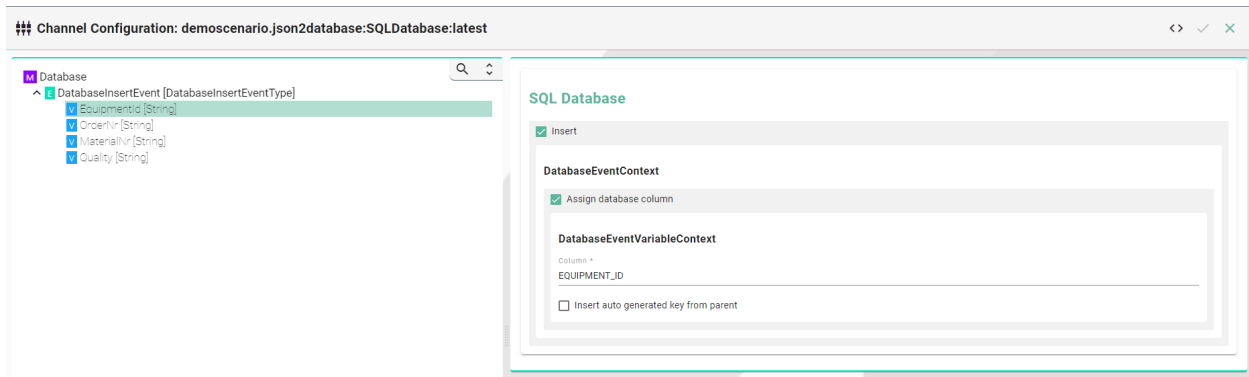


- Select the Event node in the Information Model and configure the table settings
 - Enable the checkbox *Insert*
 - Enter the name for the *Table* as well as the *Schema*



- 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*

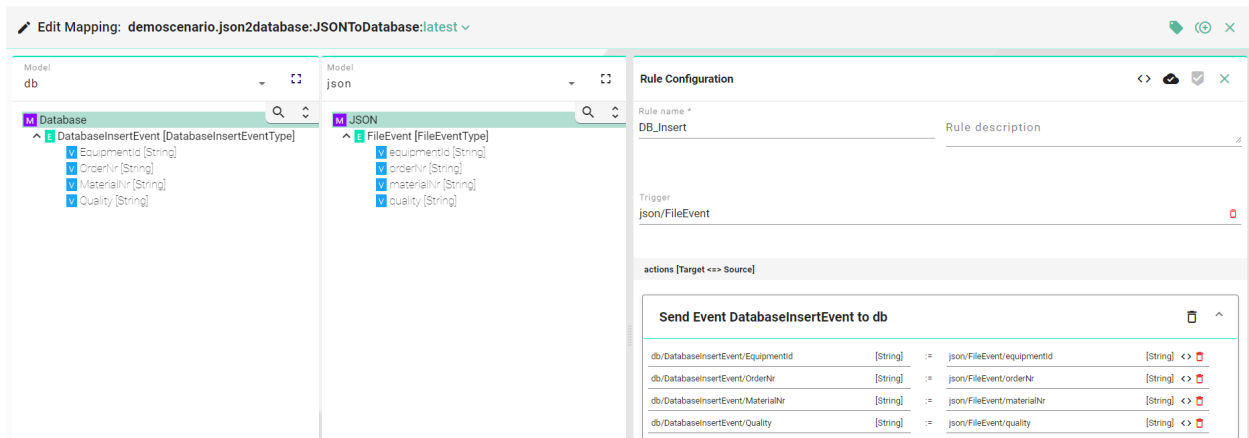


6.2.5 Mapping

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



6.2.6 Device Type

Create a new Device Type.

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

The screenshot shows the 'Edit Device Type' dialog for 'demoscenario.json2database:SUDeviceType:latest'. The dialog has a title bar with a pencil icon and the text 'Edit Device Type: democscenario.json2database:SUDeviceType:latest'. The main content area includes a 'Group' field with the value 'demoscenario.json2database', a 'Name' field with the value 'SUDeviceType', and a 'Description' field. Below these is a 'Mappings' section with a '+' icon. The 'Mappings' section contains a table with two columns: 'Models' and 'Channels'. The 'Models' column has two entries: 'db' and 'json'. The 'Channels' column has two entries: 'demoscenario.json2database:SQLDatabase:latest' and 'demoscenario.json2database:JSON:latest'. Each entry in the 'Channels' column has a gear icon to its right.

6.2.7 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

The screenshot shows the 'Edit Instance' dialog for 'demoscenario.json2database:SUIInstance:latest'. The dialog has a title bar with a pencil icon and the text 'Edit Instance: democscenario.json2database:SUIInstance:latest'. The main content area includes a 'Device Type' dropdown menu with the value 'demoscenario.json2database:SUDeviceType:latest', a 'Group' field with the value 'demoscenario.json2database', a 'Name' field with the value 'SUIInstance', and a 'Description' field. Below these is a 'Mappings' section with a '+' icon. The 'Mappings' section contains a table with two columns: 'Models' and 'Channels'. The 'Models' column has two entries: 'db' and 'json'. The 'Channels' column has two entries: 'demoscenario.json2database:SQLDatabase:latest' and 'demoscenario.json2database:JSON:latest'. Each entry in the 'Channels' column has a gear icon to its right.

6.2.8 Deployment

Create a new **Local** Deployment

- Select the Instance *UC1_Instance*
- Select the Log File Configuration *Info* (Defines the log level)

Deploy and Start the Instance *UC1_Instance*

Group	Name	Version	Deployment Type	State
demoscenario.json2database	SUInstance	latest	Local	NotDeployed

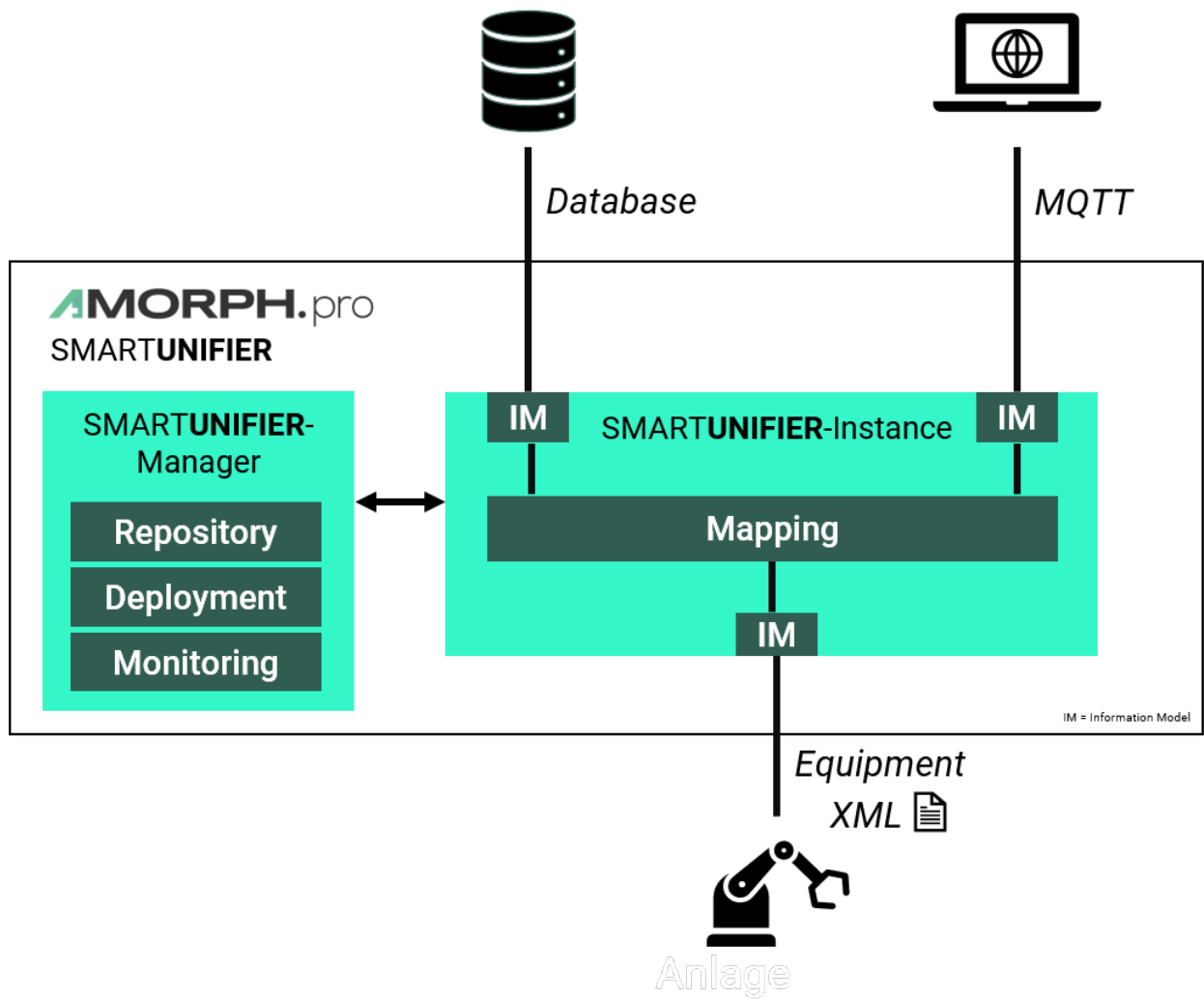
Execution

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

6.3 File-based data - XML, Database, and MQTT

6.3.1 Overview

This Scenario describes step by step how XML data can be sent 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.



6.3.2 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](#).

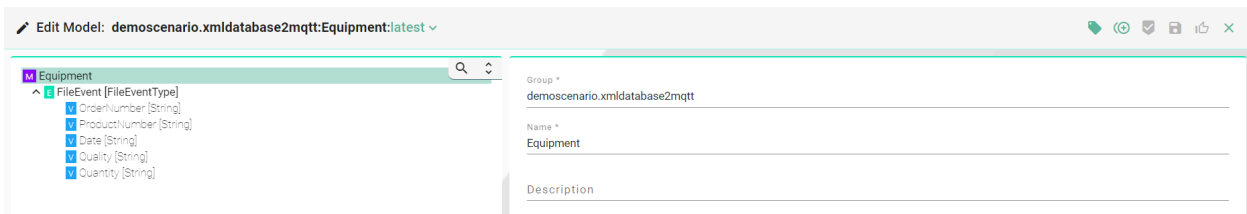
6.3.3 Information Model

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

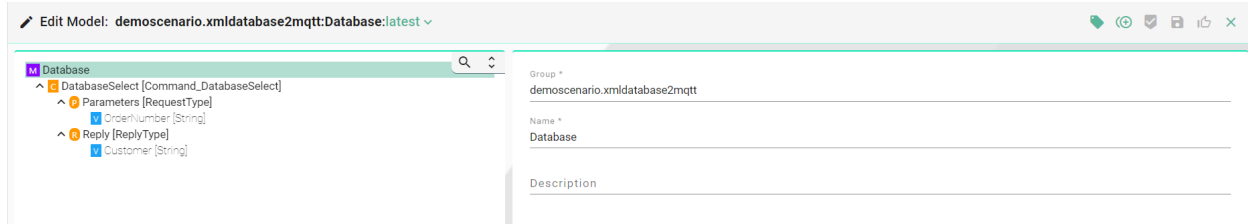


Information Model - Database

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.

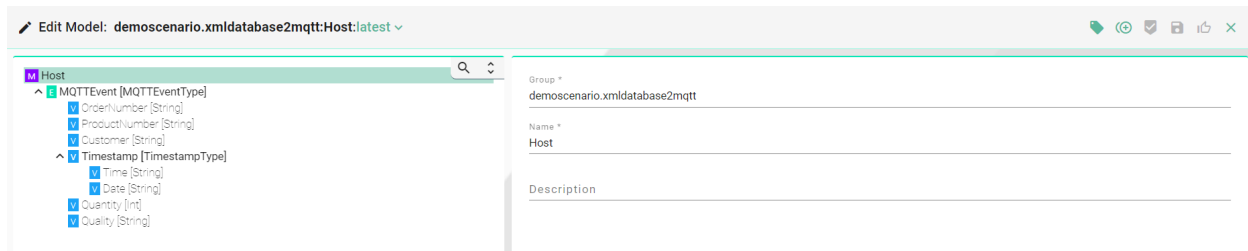


Information Model - Host (MQTT)

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



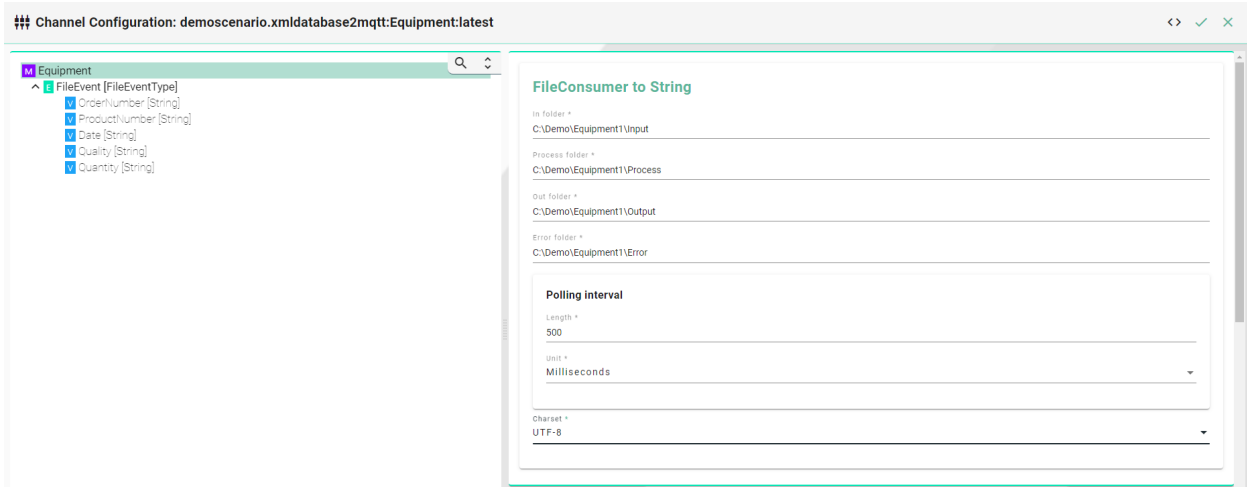
6.3.4 Communication Channel

Communication Channel - Equipment

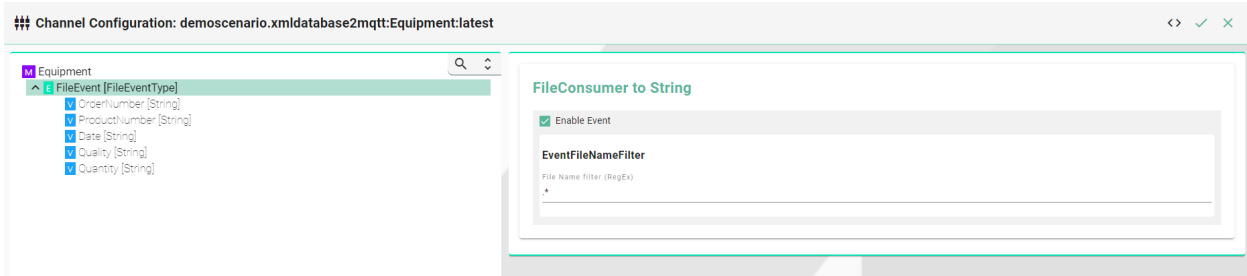
In this scenario the XML-file is processed by the SMARTUNIFIER with the build-in File Reader.

1. Create File Reader Channel:
 - Select the **Equipment** 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*



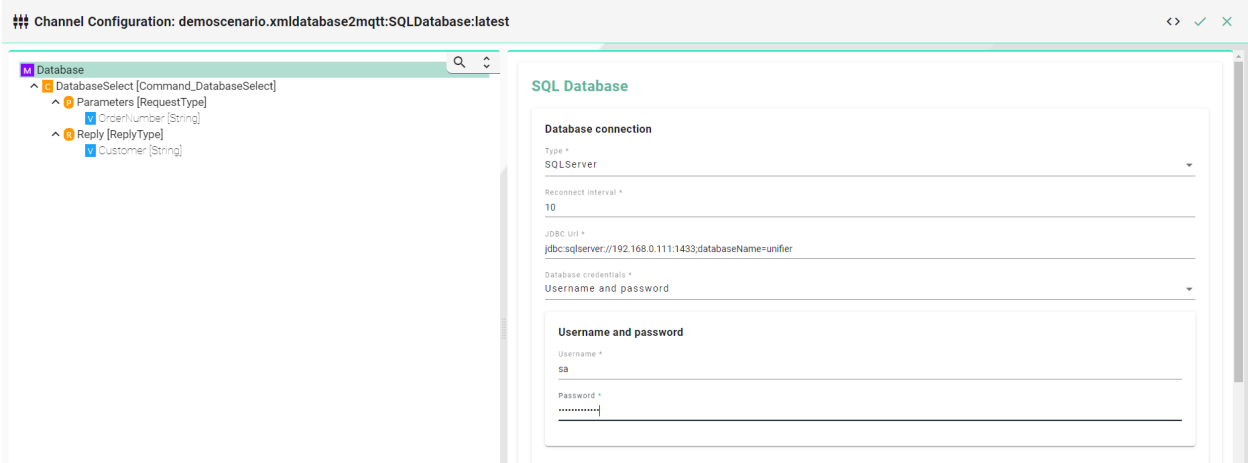
Communication Channel - SQL Database

1. Create a SQL Database Channel:

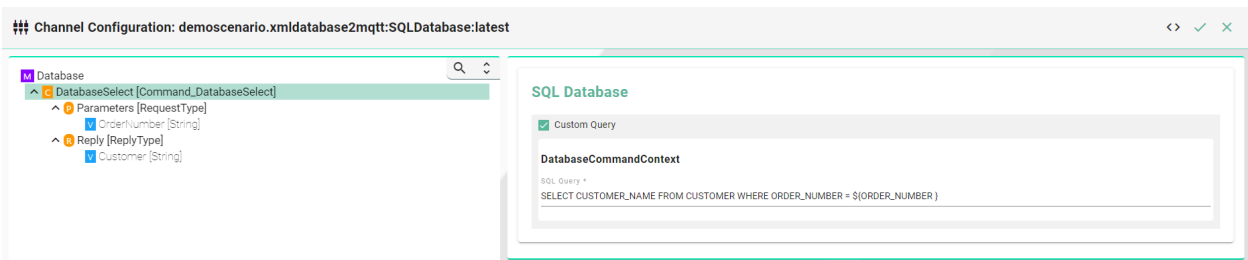
- Select the **Database** Information Model created previously.
- Select **SqlDatabase** as Channel Type.

2. Configuration:

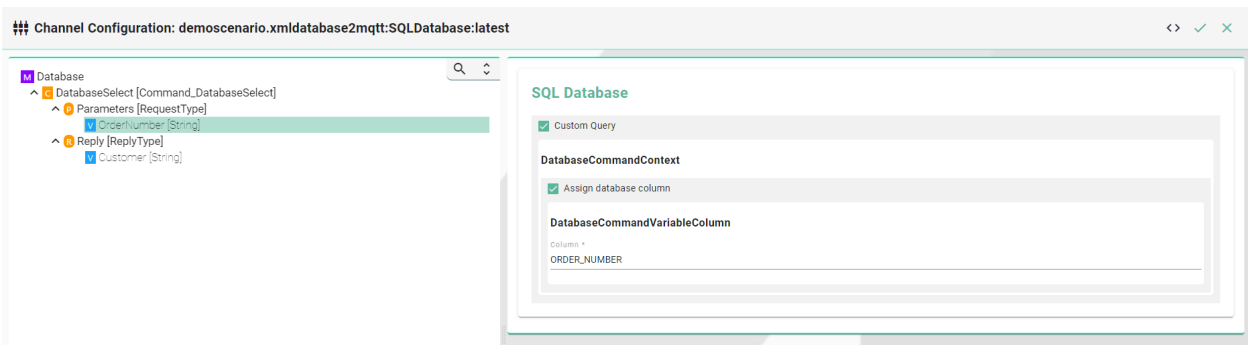
- Database Connection:
 - Select the database type **SQLServer**.
 - Set the **JDBC Url**, *according to the selected database type* - jdbc:sqlserver://192.168.0.111:1433;databaseName=unifier
 - Enter **username** and **password**.



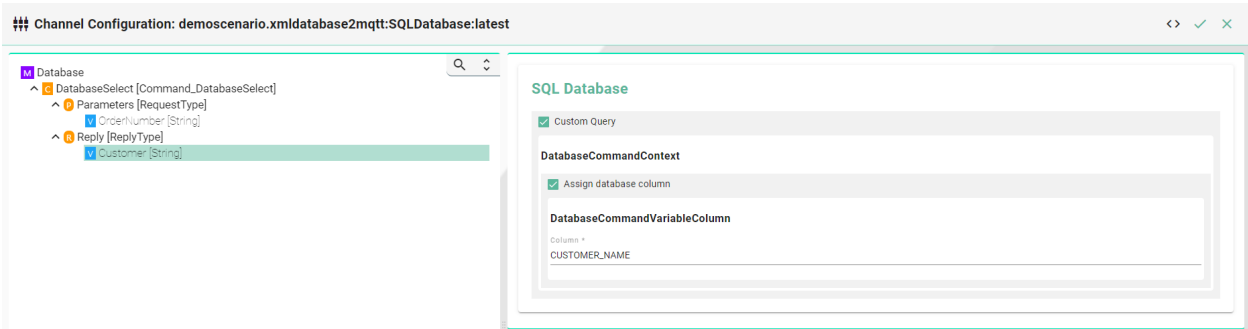
- Enter the SELECT query - select CUSTOMER_NAME from DEMO_INTEGRATION_UC3_SCHEMA.CUSTOMER where ORDER_NUMBER = \${ORDER_NUMBER}



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



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



Communication Channel - 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: demoscenario.xml:database2mqtt:Host:latest

MQTT to String

Host *
127.0.0.1

Port *
1884

Reconnect interval *
5

Connection timeout *
60

Keep alive interval *
60

Persistence folder *
persist/\${ClientId}

Client ID
1

QoS *
1

☐ Retained
Credentials
None

☒ Hostname Verification

☐ TLS Configuration

☐ Disconnected Buffer

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

Channel Configuration: demoscenario.xml:database2mqtt:Host:latest

MQTT to String

☐ Consumers

☒ Producers

TopicConfiguration

Topic *
demo/order_details

6.3.5 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.

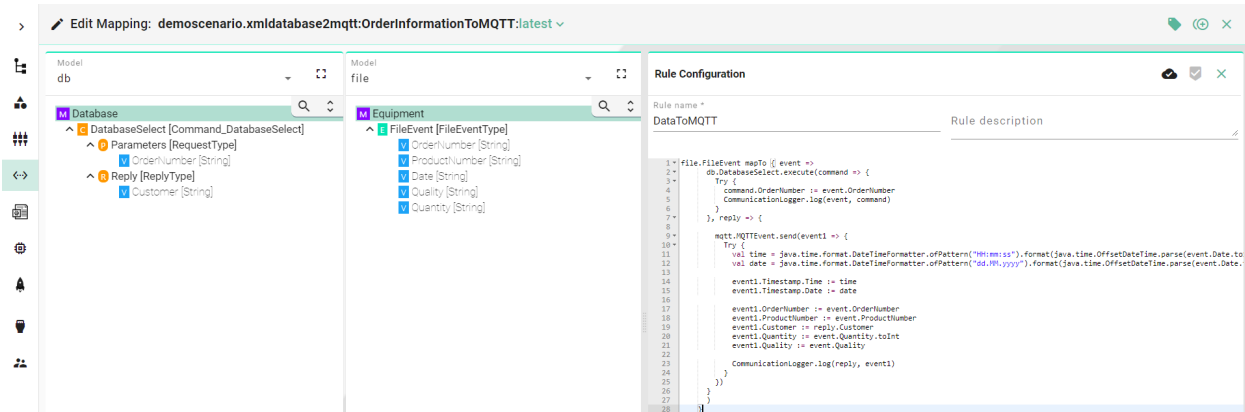
- Use the following code snippet and paste it into the Rule code editor:

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

      Try {
        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

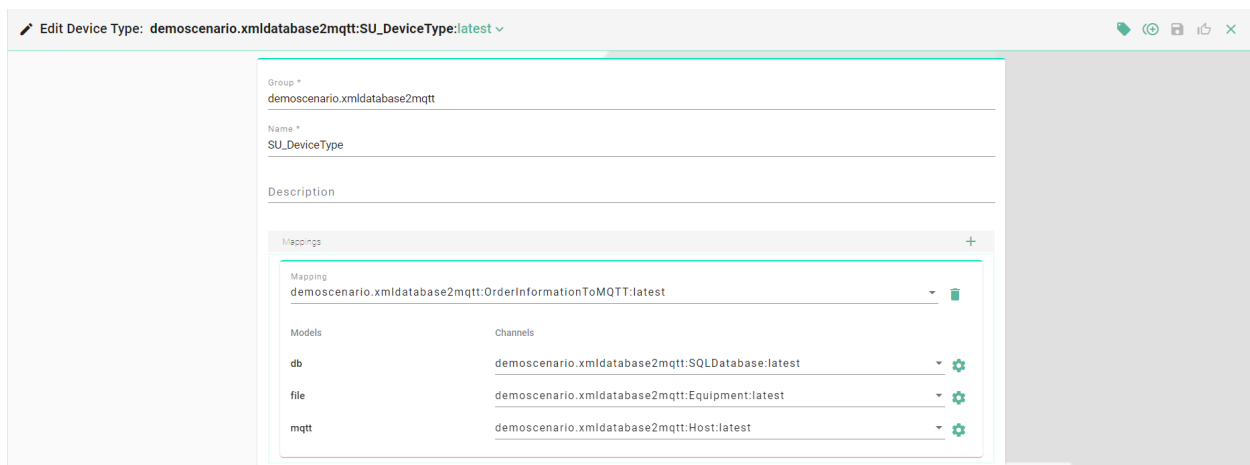
        CommunicationLogger.log(reply, event1)
      }
    }
  })
}
```



6.3.6 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.



6.3.7 Instance

Create an Instance.

- Select the *Device Type* created previously.

Edit Instance: demoscenario.xmldatabase2mqtt:SUInstance:latest

Device Type
demoscenario.xmldatabase2mqtt:SUDeviceType:latest

Group *
demoscenario.xmldatabase2mqtt

Name *
SUInstance

Description

Mappings

Mapping	Models	Channels
demoscenario.xmldatabase2mqtt:OrderInformationToMQTT:latest		
	db	demoscenario.xmldatabase2mqtt:SQLDatabase:latest
	file	demoscenario.xmldatabase2mqtt:Equipment:latest
	mqtt	demoscenario.xmldatabase2mqtt:Host:latest

6.3.8 Deployment

Create a new **Local** Deployment.

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

Add Local Deployment

Instance *
demoscenario.xmldatabase2mqtt:SUInstance:latest

Log File Configuration *
Info

Deploy and Start the Instance.

Deployment

Group Filter

<

Group

↑

Name

Version

Deployment Type

State

▼ Show All

▼ demoscenario

xmldatabase2mqtt

xmldatabase2mqtt

demo

scenario.xml

database2mqtt

SU

Instance

latest

Local

NotDeployed

▶

■

⌵

⌴

✎

🗑

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 would 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.

7.1 Troubleshooting

7.1.1 Communication Instance works abnormally or crashes

1. Determine if there is an issue with the deployment environment (VM, Cloud, other Hardware) where the Communication [Instance](#) is operated on.
2. In case of a HW problem setup a new HW (or switch to a spare HW). Ensure to place the correct security certificates on the new HW. Perform a new [deployment](#) 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.

7.1.2 Manager works abnormally or 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.

7.2 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).

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).

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

Yes, SMARTUNIFIER is specifically designed to strongly support 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 can 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 can 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 can use any file in any format 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 can receive 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 can 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 can support 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 support Windows Active Directory (AD)?

Yes, SMARTUNIFIER supports *Windows Active Directory*.

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 to 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 analysed with any analytics tools.

What is the cost model of SMARTUNIFIER ?

Please refer to Amorph Systems (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).

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).

Does SMARTUNIFIER support multiple languages?

Yes, SMARTUNIFIER can support 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)

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.

7.3 Glossary

Arrays An Array (as an Information Model Node Type) is a container object that holds a fixed number of values of a single type.

Configuration Components Configuration Components are Information Models, Communication Channels, Mappings, Device Type and Communication Instances, used to realize an integration scenario.

Commands Commands (as an Information Model Node Type) are functions like the methods of a class in object-oriented programming. The scope of a Command is bounded to the Information Model it belongs.

Communication Channels Communication Channels or simply Channels, refer 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 some form of pathway

or medium. These pathways are called Communication Channels, and the information is transmitted with the help of communication protocols. Each Information Model has one or many Channels, and each Information Model can choose which Channel 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.

Custom Types Custom Data types are defined by the user for a Node Type.

Data Types Each Node Type has a Data Type. Data Types can be either a Simple Type or a Custom Type - depending on the Node Type.

Deployments With the SMARTUNIFIER Deployment capability Instances can be deployed to any IT resource (e.g., Equipment PC, Server, Cloud) suitable to execute a SMARTUNIFIER Instance.

Deployment Endpoints Deployment Endpoints are used to identify the location of a Deployment (e.g., AWS Fargate, Docker)

Device Types 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 overwrite existing Communication Channel configurations. Device Types are especially important, when having to connect several similar equipment or devices that share the same communication parameters. In these cases it is sufficient to define only one Device Type and the settings of this Device Type can be reused across all Instances.

Events Events (as an Information Model Node Type) represent an action or occurrence recognized by SMARTUNIFIER, often originating asynchronously from an external data source (e.g., equipment, device). 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).

File Consumer This Communication Channels offers the capability to read-in files (e.g., CSV, XML, and JSON). The File Consumer monitors an input folder that is specified in the configuration.

File Tailer This Communication Channels offers the capability to read-in files (e.g., CSV, XML, and JSON). The File Tailer monitors a specific file, which is specified in the configuration.

InfluxDB This Channel offers connectivity to an InfluxDB. InfluxDB is an open-source time series database.

Information Models Information Model describes the communication related data, which is available for a device or IT system. Each device or IT system is represented by an Information Model.

Instances An Instance represents an application that handles the connectivity. Instances can be deployed to any suitable IT resource (e.g., Equipment PC, Server, Cloud), and provide the connectivity functionality configured. Therefore, a SMARTUNIFIER Instance uses one or multiple Mappings and selected Communication Channels from a previously defined Device Type.

Lists A List (as an Information Model Node Type) representing collections of Node Types (e.g., Variables, Properties, Arrays, and other Lists).

Mappings Mapping represents the SMARTUNIFIER component that is defining 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.

MQTT This Communication Channel offers the capability to send data using the messaging protocol MQTT. MQTT is a lightweight publish/subscribe messaging transport for connecting remote devices with minimal network bandwidth.

Node Types Node Types are elements within an Information Model. Node Types are Variables, Properties, Events, Commands and also collections such as Arrays and Lists. Each Node Type has a Data Type that defines if the Node Type is a predefined Data Type or a custom Data Type.

OPC-UA Is a machine to machine communication protocol for industrial automation.

Predefined Types Predefined Data Types (e.g., String, Integer, Double, etc.) are available for the definition types - Variables, Properties, Arrays, Lists (e.g., String, Integer, Boolean).

Properties Properties (as an Information Model Node Type) are used to represent XML attributes.

REST Client This Communication Channels offers the capability to consume and operate with resources exposed by REST Servers.

REST Server This Communication Channels offers the capability to provide resources employing the HTTP communication protocol.

Rules A Rule contains a Trigger that defines when the exchange/transformation takes place and a list of actions that are defining how the exchange/transformation is done.

Manager The Web application SMARTUNIFIER Manager is used to create and monitor SMARTUNIFIER Instances.

Source In the Mapping sources are Node Types that are mapped to targets.

SQL DB This Communication Channel offers the capability to connect to a SQL Databases (e.g., MariaDB, SQLServer, PostgreSQL, ORACLE, HSQLDB, and DB2).

Target In the Mapping targets are Node Types that receive data assigned from a source.

Trigger The Trigger defines when the exchange/transformation data between two or multiple Information Models takes place.

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

Variables Variables (as an Information Model Node Type) are used to represent values.