Cloud Connected Sensor Data-as-a-Service

Starter Solutions

Original instructions

EIO0000003797.01
11/2020

Telemecanique Sensors

Legal Information

Table of Contents

Safety Information

Important Information

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book

Document Scope

The purpose of this document is to provide to the Customer information about what is the Cloud Connected Sensor Data-as-a-Service offer, how it can be activated, exploited and what are the possibilities offered.

Validity Note

This document has been updated with Data-as-a Service - Starter Solutions V1.1

For product compliance and environmental information (RoHS, REACH, PEP, EOLI, etc.), go to www.se.com/ww/en/work/support/green-premium/.

The technical characteristics of the devices described in the present document also appear online. To access the information online, go to the Schneider Electric home page www.se.com/ww/en/download/.

The characteristics that are described in the present document should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the document and online information, use the online information as your reference.

Related Documents

Title of documentation	Reference number
XIOT11SE••• Standalone Transmitter - Instruction Sheet	QGH83382 (ENG) QGH83382 (ITA) QGH83382 (SPA) QGH83382 (FRE) QGH83382 (DAN) QGH83382 (GER) QGH83382 (SWE) QGH83382 (DUT)
Cloud Connected Sensor Data-as-a-Service - Starter Solution WonderWare	XIOTDaaSStarterSolution_WonderWare
Cloud Connected Sensor Data-as-a-Service - Starter Solution Visual Studio	XIOTDaaSStarterSolution_Visual Studio

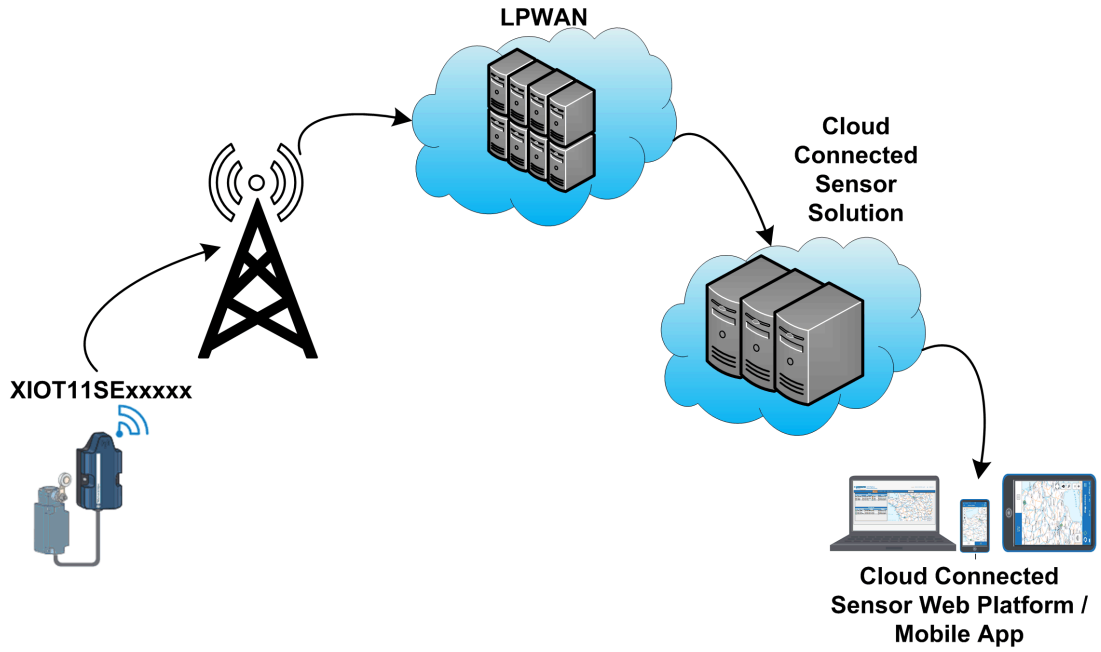
You can download these technical publications, the present document and other technical information from our website www.se.com/en/download/.

Overview of Cloud Connected Sensor Data-as-a-Service

Cloud Connected Sensor Legacy Offer

Overview

Legacy Cloud Connected Sensor overview:



The XIOT11SE••••• standalone transmitter allows to exploit state changes of one or two dry contacts through a LPWAN (Sigfox) connection. The transmitted information is made available on the web, through the Cloud Connected Sensor platform (https://XIOT.Tesensors.com) or from IOS / Android mobile applications (Cloud Connected Switch App).

NOTE: Cloud Connected Sensor Data-as-a-Service is concurrent and additive to Cloud Connected Sensor legacy features. Whatever Data-as-a-Service is activated or not, the data of the transmitter are still available on the Cloud Connected Switch Web Platform of Mobile App.

Cloud Connected Sensor Data-as-a-Service Offer

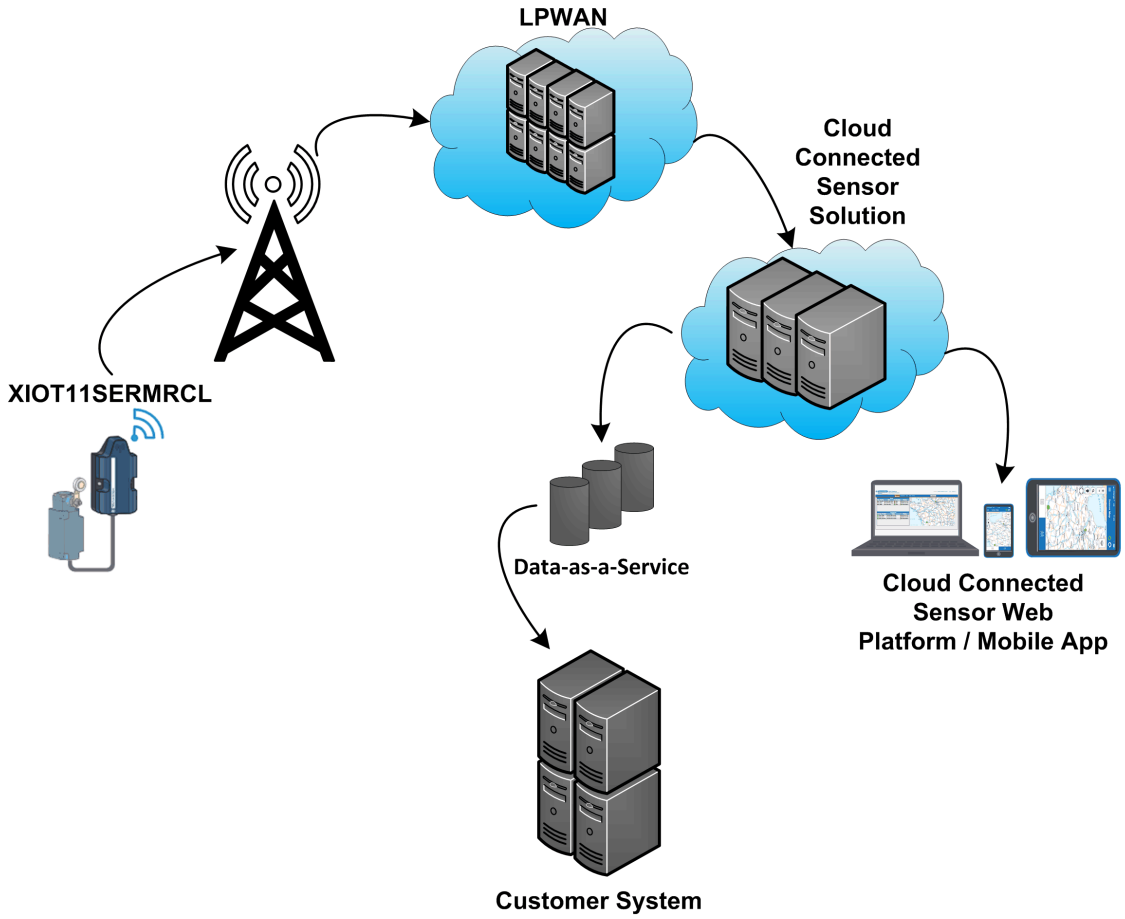
Overview

Cloud Connected Sensor Data-as-a-Service (DaaS) provides integration capability to store data and monitor Cloud Connected Sensor devices with the customer systems, such as SCADA systems or specific Customer Application systems.

NOTE: Cloud Connected Sensor Data-as-a-Service integration solution is available only for the reference XIOT11SERMRCL.

Cloud Connected Sensor propose a Near-Real-Time, Message Oriented solution for Data-as-a-Service.

Cloud Connected Sensor with Data-as-a-Service overview:



Cloud Connected Sensor Solution

An integration solution allows to extract data from the Source System and load it into the Destination System.

In the case of Cloud Connected Sensor Data-as-a-Service:

- the Source System is the Cloud Connected Sensor Solution,
- the Destination System is always the Customer System.

Cloud Connected Sensor Data-as-a-Service provides a way of retrieving information of a transmitter from the Cloud Connected Sensor platform and provide the possibility to store and monitor the data in the customer systems, whatever it is.

These processes are commonly referred to as **ETL**:

- **E** means **Extract** Data from Source System into Extract Format.
- **T** means **Transform** Data from Extract Format to Load Format.
- **L** means **Load** Data into Destination System

The Publish / Subscribe Messaging principle is used to make the data available to the Destination System.

According to the ETL concept, the action of retrieving data from the Source System is called Extract Process. However, in Publish / Subscribe Messaging concept, the action of retrieving data from the data storage is called Consume Process.

In the following chapters, the terminology Extract / Consume refers to the same action of retrieving data from a Data Source. As the core concept of Cloud Connected Sensor Data-as-a-Service is Publish / Subscribe Messaging, the terminology Consume Process is preferred.

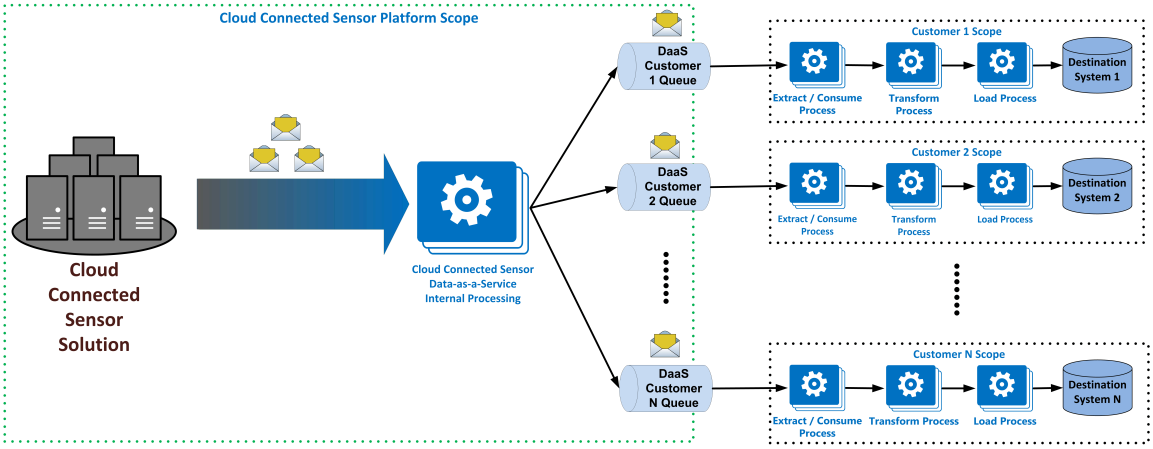
For more details on message-oriented concepts, refer to Publish / Subscribe Messaging Principle and ETL Processes, page 36.

Publish / Subscribe Messaging and ETL Processes

Publish / Subscribe Messaging and ETL Processes Applied to Cloud Connected Sensor Data-as-a-Service

Overview

Conceptual view of publish / subscribe messaging and ETL processes applied to the Cloud Connected Sensor Data-as-a-Service solution:



Principle

In Cloud Connected Sensor Data-as-a-Service case:

- The Cloud Connected Sensor Platform is the Message Publisher.
- The DaaS Customer Queue is the Publish / Subscribe Platform based on Microsoft Azure Service Bus Queue technology.
- The Communication Mechanisms are the Microsoft Azure Service Bus API (application program interface).
- The Message Consumer is the Customer who activated Cloud Connected Sensor Data-as-a-Service.
- This is the Message Consumer who implements the Extract (Consume) Process, Transform Process and Load Process.

After the extract process is executed, the data are loaded in the destination system and are no longer stored in the Cloud Connected Sensor Platform.

⚠WARNING

POTENTIAL COMPROMISE OF SYSTEM AVAILABILITY, INTEGRITY AND CONFIDENTIALITY

- Place networked devices behind multiple layers of cyber defenses (such as firewalls, network segmentation, and network intrusion detection and protection).
- Use cybersecurity best practices (for example, least privilege, separation of duties) to help prevent unauthorized exposure, loss, modification of data and logs, or interruption of services.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Activating Data-as-a-Service from the Cloud Connected Sensor Platform gives access to the Connection String.

A Connection String is a string that specifies information about a Data Source and the means of connecting to it. In the Cloud Connected Sensor Data-as-a-Service case, the provided Connection String is for a Microsoft Azure Service Bus Queue.

For more information about how to use the provided Connection String, refer to the Starter Solutions, and especially to the Consume function, page 24.

Additional information can be found on the Microsoft Azure website about how to use a Connection String and how to connect to a Microsoft Azure Service Bus Queue using dedicated API, refer to information about technologies used, page 41.

Data-as-a-Service in Cloud Connected Sensor

How to Enable Data-as-a-Service in Cloud Connected Sensor

Functionalities linked to Cloud Connected Sensor Data-as-a-Service

Overview

Cloud Connected Sensor Data-as-a-Service is available only for the reference XIOT11SERMRCL. The **Services** page of the Cloud Connected Sensor Web Platform is accessible only if an XIOT11SERMRCL device has been claimed.

The **Services** page of the Cloud Connected Sensor Web Platform permits to:

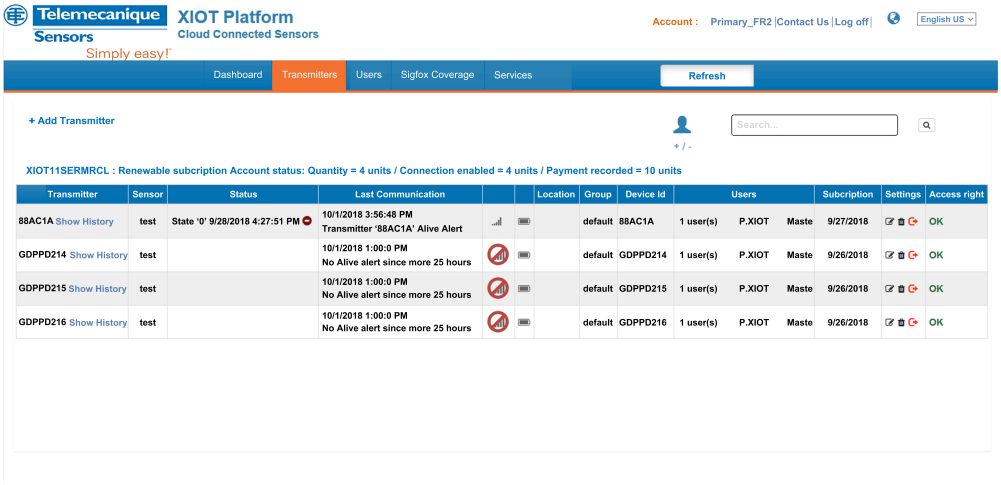
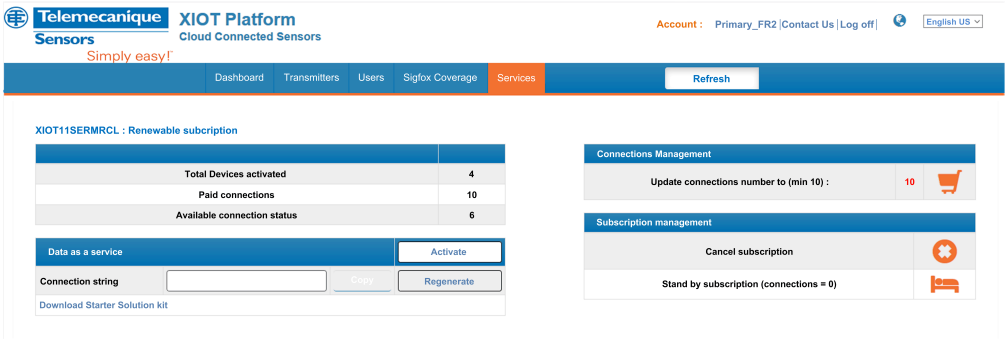
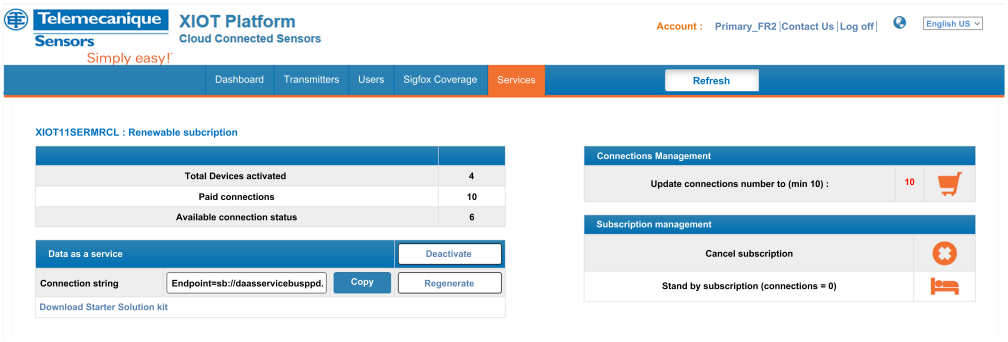
- Buy subscriptions (connections) for the XIOT11SERMRCL devices, using the **Cart** icon.
- Suspend or abort all subscriptions for devices.
- Activate / Deactivate Data-as-a-Service.
- Regenerate the Connection String.
- Access the Starter Solutions files.

Cloud Connected Sensor Data-as-a-Service Behavior

Information about Cloud Connected Sensor Data-as-a-Service behavior:

- Once Data-as-a-Service is activated, all the data coming from XIOT11SERMRCL device with a valid subscription are available in near real time in the Customer Queue.
- In the case where, Cloud Connected Sensor Data-as-a-Service is already Activated, and the Customer Claim a new Transmitter, a setup delay of 15 min is to be planned before data coming from this device are available in the Customer Queue.
- All the data in the Customer Queue will be available for 15 days.
- Once the Customer connects to his Queue and Consumes a message, this message is not available in the Queue anymore.
- In the DaaS Customer Queue, FIFO (First In First Out) order is respected.
- If the connection to the Queue (DaaS Queue) has been compromised, it is possible to Regenerate his Connection String, using the dedicated button on the **Services** page. The previous Connection String will be invalidated, and the Customer Queue will be accessible only with the new Connection String. All the data present in the Queue when the Connection String was regenerated are kept.
- Only the messages coming from a device for which the Customer is the Master will be forwarded in the Customer Queue. Thus, no data from devices for which the Customer is Admin or User will be accessible in the Customer Queue.

Step by Step Guide to Activate Cloud Connected Sensor Data-as-a-Service

Step	Action
1	Once an XIOT11SERMRCL is claimed, **Services** tab on the Cloud Connected Sensor platform becomes accessible. Cloud Connected Sensor Data-as-a-Service is managed from this page. 
2	In order to activate Cloud Connected Sensor DaaS,click **Activate** button. 
3	Cloud Connected Sensor DaaS is activated, and the Connection String (connection information) needed to access the Customer Queue is available in the dedicated box. 
4	Data for the transmitter is accessible in the DaaS Queue using the Connection String provided.

Messages Format of Transmitters

Messages Format

Overview

The Transmitters' data are stored in the Customer Queue using a JSON format.

Different JSON messages can be present in the DaaS Customer Queue. Each JSON message has common attributes and they differ from each other only from a few attributes.

The `RecordType` JSON attribute allows to know which JSON template is used.

Cloud Connected Sensor transmitters can send 5 types of data frame:

- A product configuration frame, page 17
- A network configuration frame, page 18
- A keep alive frame, page 19
- A measurement frame, page 20
- A geolocation frame, page 21

Timing Diagram

Frames transmitted by the transmitter:

Step	Action	Frames sent
1	Product activation after magnet detection	Product configuration frame 10_h Network configuration frame 20_h Measurement frame 40_h
2	State change of one of the two inputs	Measurement frame 40_h
3	State change of one of the two inputs during transmission of the previous frame **NOTE:** 20 s guard time between two transmissions	Measurement frame 40_h
4	One time per day	Keep alive frame 30_h

NOTE: The transmitted frames are in Little endian format

NOTE: A Geolocation frame FF_h is generated by the Sigfox Backend each time a transmitter message is received.

Interpretation of the JSON Messages

The following table presents information about how to interpret each field of the JSON messages:

JSON Field	Description	Value	Frame Type				
			10_h	20_h	30_h	40_h	FF_h
`device`	Sigfox unique device ID (Transmitter unique ID).	4 bytes			✓		
`time`	GMT timestamp (reception date).	YYYY-MM-DD**T**HH:MM:SS			✓		-
`avgSnr`	Average signal to noise ratio computed from the last 25 messages. **NOTE:** The device must have sent at least 15 messages.	In dB - Float value with two maximum fraction digits or N/A. Not provided anymore by Sigfox (always " null").			✓		-
`data`	Reserved	-			✓		-
`duplicate`	Reserved	-			✓		-
`snr`	Signal to noise ratio.	In dB - Float value with two maximum fraction digits.			✓		-
`station`	Sigfox Base Station Identifier.	2 bytes			✓		-
`lat`	Latitude, rounded to the nearest integer, of the base station which received the message.	In Deg - with one fraction digit. Not provided anymore by Sigfox (always " null").			✓		-
`lng`	Longitude, rounded to the nearest integer, of the base station which received the message.	In Deg - with one fraction digit. Not provided anymore by Sigfox (always " null").			✓		-
`rssi`	Received Signal Strength.	In dBm - Float value with two maximum fraction digits. If there is no data to be returned, then the value is null.			✓		-
`seqNumber`	Sequence numbering of the frames if available.	-			✓		-
`RecordType`	Type of the frame sent by the Cloud Connected Sensor Transmitter. • Product Configuration Frame. • Network Configuration Frame. • Keep Alive Frame. • Measurement Frame. • Geolocation Frame.	• 16 (10_h) • 32 (20_h) • 48 (30_h) • 64 (40_h) • 255(FF_h)			✓		
`FrameCnt1`	Numbering of the frames sent by the transmitter. Incremented at each transmission. Used to know if a frame has been lost.	0…7 and loop again			✓		-
`CommandDone`	Reserved	-			✓		-
`HWError`	Reserved	-			✓		-
`LowBatError`	Reserved	-			✓		-
`ConfigOK`	Reserved	-			✓		-
`S1ClosedCnt`	Number of times the Input 1 switched to state Closed since its last transmission to Cloud Connected Sensor platform.	0…255		-		✓	-
`S2ClosedCnt`	Number of times the Input 2 switched to state Closed since its last transmission to Cloud Connected Sensor platform.	0…255		-		✓	-
`S3ClosedCnt`	Number of times the Input 3 switched to state Closed since its last transmission to Cloud Connected Sensor platform.	0…255		-		✓	-
`S4ClosedCnt`	Number of times the Input 4 switched to state Closed since its last transmission to Cloud Connected Sensor platform.	0…255		-		✓	-

JSON Field	Description	Value	Frame Type				
			10_h	20_h	30_h	40_h	FF_h
`S4PreviousState`	Previous state of the Input 4 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S4State`	Current state of the Input 4 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S3PreviousState`	Previous state of the Input 3 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S3State`	Current state of the Input 3 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S2PreviousState`	Previous state of the Input 2 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S2State`	Current state of the Input 2 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S1PreviousState`	Previous state of the Input 1 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S1State`	Current state of the Input 1 connected to the Cloud Connected Sensor transmitter.	1: Open 0: Closed	-			✓	-
`S202`	Reserved	Fixed value (2).	-	✓	-		
`S300`	Reserved	Fixed value (144).	✓	-			
`S301`	Reserved	Fixed value (72).	✓	-			
`S302`	Reserved	Fixed value (70).	✓	-			
`S303`	Reserved	Fixed value (70).	✓	-			
`S304`	Reserved	Fixed value (70).	✓	-			
`S305`	Reserved	Fixed value (70).	✓	-			
`S306`	Reserved	Fixed value (1).	✓	-			
`FrameCnt2`	Reserved	Same value as `FrameCnt1`.	✓				-
`FrameCnt3`	Reserved	Same value as `FrameCnt1`.	✓				-
`FrameCnt4`	Reserved	Same value as `FrameCnt1`.	✓				-
`S1OpenCnt`	Number of times Input 1 switched to state Open since its last transmission to Cloud Connected Sensor platform.	0…255	-			✓	-
`S2OpenCnt`	Number of times Input 2 switched to state Open since its last transmission to Cloud Connected Sensor platform.	0…255	-			✓	-
`lat`	Latitude of the transmitter which transmitted the frame.	In Deg - 32 bits float value.	-				✓
`lng`	Longitude of the transmitter which transmitted the frame.	In Deg - 32 bits float value.	-				✓

Product Configuration Frame (`RecordType` = 16 (10_h))

Overview

For a Production Configuration Frame, `RecordType` equals 16 (10_h) in the JSON message.

Product Configuration Frames are sent by the Cloud Connected Sensor transmitter only after its activation.

Receiving such a frame during normal operation of the product can be interpreted as a sign of abnormal behavior.

Example of a Product Configuration Frame


```
{
  "device": "88536A",
  "time": "2018-10-04T13:39:30",
  "avgSnr": "null",
  "data": "100090484646464601",
  "duplicate": "false",
  "snr": "6.00",
  "station": "0FBE",
  "lat": "null",
  "lng": "null",
  "rssi": "-135.00",
  "seqNumber": "1254",
  "RecordType": 16,
  "FrameCnt1": 0,
  "CommandDone": 0,
  "HWError": 0,
  "LowBatError": 0,
  "ConfigOK": 0,
  "S300": 144,
  "S301": 72,
  "S302": 70,
  "S303": 70,
  "S304": 70,
  "S305": 70,
  "S306": 1,
  "FrameCnt2": 0,
  "FrameCnt3": 0,
  "FrameCnt4": 0,
  "SwitchError": 0,
}
```

Network Configuration Frame (`RecordType` = 32 (20_h))

Overview

For a Network Configuration Frame, `RecordType` equals 32 (20_h) in the JSON message.

Network Configuration Frames are sent by the Cloud Connected Sensor transmitter only after its activation.

Receiving such a frame during normal operation of the product can be interpreted as a sign of abnormal behavior.

Example of a Network Configuration Frame


```
{
  "device": "88536A",
  "time": "2018-10-04T13:40:06",
  "avgSnr": "null",
  "data": "202002",
  "duplicate": "false",
  "snr": "8.40",
  "station": "0FBE",
  "lat": "null",
  "lng": "null",
  "rssi": "-136.00",
  "seqNumber": "1255",
  "RecordType": 32,
  "FrameCnt1": 1,
  "CommandDone": 0,
  "HWError": 0,
  "LowBatError": 0,
  "ConfigOK": 0,
  "S202": 2,
  "FrameCnt2": 1,
  "FrameCnt3": 1,
  "FrameCnt4": 1,
  "SwitchError": 0
}
```

Keep Alive Frame (`RecordType` = 48 (30_h))

Overview

For a Keep Alive Frame, `RecordType` equals 48 (30_h) in the JSON message.

Keep Alive Frames are sent by the Cloud Connected Sensor transmitter every 24 hours.

Example of a Network Configuration Frame


```
{
  "device": "88AC1A",
  "time": "2018-09-28T13:57:01",
  "avgSnr": "null",
  "data": "3060",
  "duplicate": "false",
  "snr": "13.21",
  "station": "05A1",
  "lat": "null",
  "lng": "null",
  "rssi": "-135.00",
  "seqNumber": "723",
  "RecordType": 48,
  "FrameCnt1": 3,
  "CommandDone": 0,
  "HWError": 0,
  "LowBatError": 0,
  "ConfigOK": 0,
  "FrameCnt2": 3,
  "FrameCnt3": 3,
  "FrameCnt4": 3,
  "SwitchError": 0
}
```

Measurement Frame (`RecordType` = 64 (40_h))

Overview

For a Measurement Frame, `RecordType` equals 64 (40_h) in the JSON message.

Measurement Frames are sent by the Cloud Connected Sensor each time the output state of the sensor is modified.

Example of a Network Configuration Frame


```
{
  "device": "88AC1A",
  "time": "2018-09-28T13:28:38",
  "avgSnr": "null",
  "data": "40800001000000000000001",
  "duplicate": "false",
  "snr": "8.20",
  "station": "05A1",
  "lat": "null",
  "lng": "null",
  "rssi": "-137.00",
  "seqNumber": "716",
  "RecordType": 64,
  "FrameCnt1": 4,
  "CommandDone": 0,
  "HWError": 0,
  "LowBatError": 0,
  "ConfigOK": 0,
  "S1ClosedCnt": 1,
  "S2ClosedCnt": 0,
  "S3ClosedCnt": 0,
  "S4ClosedCnt": 0,
  "S4PreviousState": 0,
  "S4State": 0,
  "S3PreviousState": 0,
  "S3State": 0,
  "S2PreviousState": 0,
  "S2State": 0,
  "S1PreviousState": 0,
  "S1State": 0,
  "FrameCnt2": 4,
  "FrameCnt3": 4,
  "FrameCnt4": 4,
  "SwitchError": 0
  "S1OpenCnt": 1,
  "S2OpenCnt": 0,
}
```

Geolocation Frame (RecordType = 255 (FFh))

Overview

For a Geolocation Frame, `RecordType` equals 255 (FF_h) in the JSON message.

Geolocation Frames are generated by the Sigfox Backend each time a transmitter message is received

Example of a Geolocation Frame

```
{
  "device": "88AC1A",
  "RecordType": 255,
  "lat": "46.751289126",
  "lng": "1.0254862135",
}
```

Radio Data Transmission

Radio Frames Details

Overview

The transmitter sends four frame types on the SIGFOX® network:

- A product activation frame (code 10_h).
- A network activation frame (code 20_h).
- A keep alive frame (code 30_h).
- An event frame for sensor inputs (code 40_h).

Transmitted Product Configuration Frame Detail

Product configuration `Data` frame 10_h:

Byte number		Data	Description
0		Record Type Code	10_h (16)
1	Bit 0	Status	00_h
	Bit 1		= 1 if the battery voltage is ≤2.5 Vdc
	Bit 2		= 1 if an hardware error has been detected
	Bit 3…4		Reserved
	Bit 5…7		Transmitted frames counter (00_h…07_h)
2		-	90_h
3		-	48_h
4		-	46_h
5		-	46_h
6		-	00_h
7		-	00_h
8		-	01_h
9		-	00_h
10		-	00_h

Transmitted Network Configuration Frame Detail

Network configuration `Data` frame 20_h:

Byte number		Data	Description
0		Record Type Code	20_h (32)
1	Bit 0	Status	00_h
	Bit 1		= 1 if the battery voltage is ≤ 2.5 Vdc
	Bit 2		= 1 if an hardware error has been detected
	Bit 3…4		Reserved
	Bit 5…7		Transmitted frames counter (00_h…07_h)
2		-	01_h
3		-	01_h
4		-	00_h
5		-	00_h
6		-	00_h
7		-	00_h

Byte number	Data	Description
8	-	00_h
9	-	00_h
10	-	00_h

Transmitted Keep Alive Frame Detail

Keep alive `Data` frame 30_h:

Byte number		Data	Description
0		Record Type Code	30_h (48)
1	Bit 0	Status	00_h
	Bit 1		= 1 if the battery voltage is ≤ 2.5 Vdc
	Bit 2		= 1 if an hardware error has been detected
	Bit 3…4		Reserved
	Bit 5…7		Transmitted frames counter (00_h…07_h)

Transmitted Measurement Frame Detail

Measurement `Data` frame 40_h:

Byte number		Data	Description
0		Record Type Code	40_h (64)
1	Bit 0	Status	00_h
	Bit 1		= 1 if the battery voltage is ≤ 2.5 Vdc
	Bit 2		= 1 if an hardware error has been detected
	Bit 3…4		Reserved
	Bit 5…7		Transmitted frames counter (00_h…07_h)
2…3		Sensor 1	Event counter (00_h…FF_h)
4…5		Sensor 2	Event counter (00_h…FF_h)
6…9		Reserved	-
10	Bit 0	Sensor State 1: Open, 2: Closed	Sensor 1 - Current state
	Bit 1		Sensor 1 - State at previous frame
	Bit 2		Sensor 2 - Current state
	Bit 3		Sensor 2 - State at previous frame
	Bit 4		Sensor 3 - Current state
	Bit 5		Sensor 3 - State at previous frame
	Bit 6		Sensor 4 - Current state
	Bit 7		Sensor 4 - State at previous frame

Cloud Connected Sensor Data-as-a-Service Starter Solutions

Overview

The Starter Solutions are documented software source code provided as a .NET Core console application, that provides examples of how to implement Publish / Subscribe Messaging and Extract, Transform, Load processes.

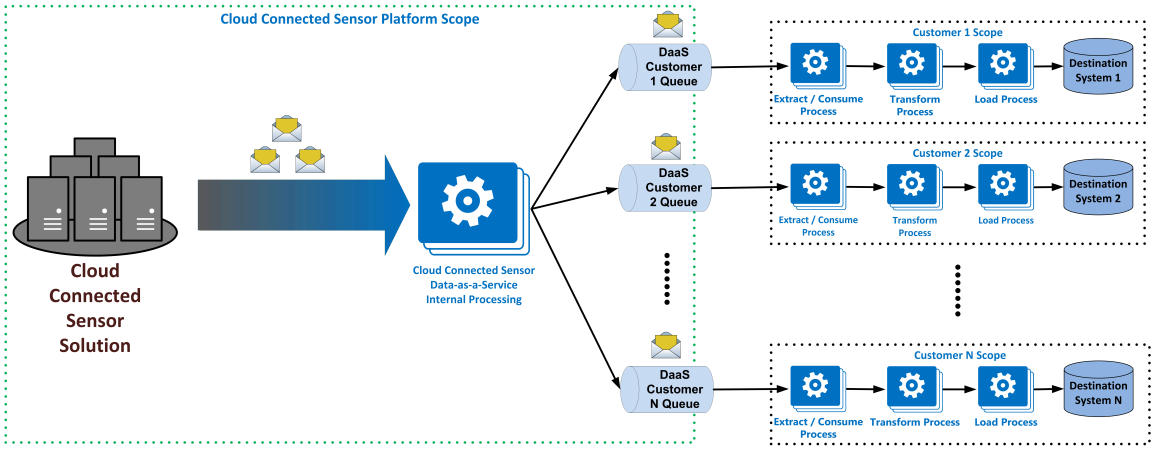
Two Destination Systems are covered by the Starter Solutions:

- A simple File stored in a specific Windows folder
- WonderWare Historian (not to be confused with WonderWare Online)
- Modbus over TCP

Conceptual View of the Starter Solutions

Overview

Conceptual view of the Starter Solutions for Cloud Connected Sensor Data-as-a-Service integration:



The Starter Solutions help to understand how each process operates and is implemented. The Starter Solutions can be customized to Consume, Transform and Load data into the Historian.

From an architecture point of view, all the Starter Solutions are architected in the same way:

- A first part, dedicated to setups, in which the connection to the Source System (the DaaS Customer Queue) and the Destination System is made.
- A core part, in which the 3 ETL (Extract, Transform, Load) processes are implemented.
- A last part, in which disconnection from the Source System and the Destination System is made.

Depending on the Destination System, the Transform Process may be optional. Transform Process is always present in the Starter Solutions, but it may be a null operator.

All Starter Solutions are provided as a standalone Microsoft Visual Studio Solution, written in C# language for a .NET Core or .NET framework. .NET Core (v2.0) and .NET Framework (v4.6) may be mandatory to run the Starter Solutions. These Packages can be installed using Microsoft Visual Studio Installer.

Before running any Starter Solution, it is mandatory to do few modifications of the source code. According each Starter Solution, information are provided about what is expected to be modified.

Consume Function

This function is responsible for Subscribing to the Customer Queue and Consuming messages from it. Subscribing to the Queue requires that the Customer has information about the Queue including Endpoint and Read security credentials. This information is provided in the Connection String.

The Consume Message function is the same code across Customers. The function differs between Customers only in the Customer unique Connection String.

Transform Function

This function is responsible for extracting the Payload from the Message received from Consume Message and converting the Payload into a format which is usable by the Load function.

For example, this function will convert the JSON Payload into an intermediate data set format which appropriate for loading into the Customer's Historian. The data set is then used by the Load Data function and loaded into the Customer Historian.

Load Function

This function is responsible for taking the data set received from the Transform function and inserting it into the Customer Destination System. Load Process is the most specialized Process in the system because it must implement esoteric Destination System data insertion logic using native libraries and classes.

Starter Solution for a Simple File

Overview

This Starter Solution is accessible in the .zip file `XIOTDaaSStarterSolution_Visual Studio.zip` and provides an example of how to Consume, Transform and Load data into a JSON File on a Windows System.

Required NuGet Packages, SDK and Libraries

Overview

This Starter Solution is provided as a .NET Core application.

Thus, only two packages have to be installed. It is the Microsoft.Azure.ServiceBus (v3.1.0) and Microsoft.NET Core.App (v2.0).

Pre-Run Setup

Overview

Before running this Starter Solution, the value of two constant locals must be modified:

- `ServiceBusConnectionString` must be updated with the Customer Connection String (available on Cloud Connected Sensor web site, on the **Services** page).

  For example:

  ```
  const string ServiceBusConnectionString = "Endpoint=
  sb://dass-test.servicebus.windows.net/;
  SharedAccessKeyName=RootManageSharedAccessKey;
  SharedAccessKey=slVnJLa7QXMm8qEc9/pGm3xtrs1wKzf
  +q3aImwq1iZI=;EntityPath=dass-queue1";
  ```

- `FilePath` must be updated with the path of the file in which the data messages will be stored.

  For example:

  ```
  const string FilePath = "C:\\Users\\Public\\DaaSTest.
  json";
  ```

Behavior of the Starter Solution

Overview

The `XIOTDaaSStarterSolution_Visual Studio` will do the following:

Step	Action
1	Connect to an Azure Service Bus Queue (the DaaS Customer Queue) using the Connection String provided in the constant local `ServiceBusConnectionString`.
2	Consume messages from the DaaS Customer Queue.
3	Display the received messages in the Console.
4	Store the received messages as is (JSON format) in a specific folder defined by the constant local `FilePath`.
5	Clear the messages from the queue.
6	Close the connection to the DaaS Customer Queue.

This Starter Solution is expected to be used as following:

Step	Action
1	Check that Data-as-a-Service is activated on Cloud Connected Sensor web platform.
2	Set up the Starter Solution by updating constant locals (`ServiceBusConnectionString` and `FilePath`).
3	Trigger message sending by switching one of the two dry contacts connected to the XIOT11SERMRCL transmitter.
4	Run the Starter Solution.
5	The messages sent by the transmitter are displayed in a Console window. The messages sent by the transmitter are also available in the file referenced by `FilePath`.

Starter Solution for WonderWare Historian

Overview

This Starter Solution is accessible in the zip file `XIOTDaaSStarterSolution_WonderWare.zip` and provides an example of how to Consume, Transform and Load data into a WonderWare Historian system.

Required NuGet Packages, SDK and Libraries

Overview

This Starter Solution is provided as a .NET Framework application since it uses third-party NuGet Package for WonderWare Historian.

For this Starter Solution, two packages must be installed

- Microsoft.Azure.ServiceBus (v3.1.0) and .Net Framework (v4.7.1)
- WonderWare Historian SDK 2017 (v2.0)

In addition, a WonderWare Historian Server 2017 must be available.

Pre-Run Setup

Overview

Before running this Starter Solution, the following elements must be modified:

- `ServiceBusConnectionString` must be updated with the Customer Connection String (available on Cloud Connected Sensor web site, on the **Services** page).

  For example:

  ```
  const string ServiceBusConnectionString = "Endpoint=
  sb://dass-test.servicebus.windows.net/;
  SharedAccessKeyName=RootManageSharedAccessKey;
  SharedAccessKey=slVnJLa7QXMm8qEc9/pGm3xtrs1wKzf
  +q3aImwq1iZI=;EntityPath=dass-queue1";
  ```

- In the `setUpWonderwareHistorianConnection()` method, the Historian Credentials must be updated:
  - `connectionArgs.ServerName` must take the value of the Historian Server Name.
  - `connectionArgs.UserName` and `connectionArgs.Password`, must be updated with the User Credentials used to connect the WonderWare Historian.

Behavior of the Starter Solution

Overview

The `XIOTDaaSStarterSolution_File` will do the following:

Step	Action
1	Connect to an Azure Service Bus Queue (the DaaS Customer Queue) using the Connection String provided in the constant local `ServiceBusConnectionString`.
2	Connect to the WonderWare Historian Server using the connection information provided (`connectionArgs.ServerName`, `connectionArgs.UserName`, `connectionArgs.Password`).
3	Consume messages from the DaaS Customer Queue.
4	Display the received messages in the Console.
5	Store the received messages as is (JSON format) in the Historian Server using Tag Format.
6	Clear the messages from the queue.
7	Close the connection to the DaaS Customer Queue and to the Historian Server.

This Starter Solution is expected to be used as following:

Step	Action
1	Check that Data-as-a-Service is activated on Cloud Connected Sensor web platform.
2	Set up the Starter Solution by updating connection information for the DaaS Customer Queue and the WonderWare Historian.).
3	Trigger message sending by switching one of the two dry contacts connected to the XIOT11SERMRCL transmitter.
4	Run the Starter Solution.
5	The messages sent by the transmitter are displayed in a Console window. The messages sent by the transmitter are also available in the WonderWare Historian.

Code Customization

The Starter Solution for WonderWare Historian is provided as a basic example. The code must be adapted to the use case / environment.

For example, the WonderWare Tags management and Tags extended properties may be modified.

NOTE: This Starter Solution has been implemented to parse only Measurement frames retrieved from the DaaS Customer Queue. To parse other frames, the existing code must be modified or an other methodology must be implemented.

Starter Solution for Modbus over TCP

Overview

This Starter Solution is accessible in a zip file and provides an example of how to Consume, Transform, and Load data into a Modbus TCP server that runs locally.

Required NuGet Packages, SDK, and Libraries

Overview

The Starter Solution is provided as a .NET Framework application since it uses third-party NuGet Package for Modbus TCP Server.

2 Starter Solution zip files are available:

- App Starter Solution: `XIOT_DaaS_SS_ModbusTCP_App_vx-y.zip`
- Full Starter Solution: `XIOT_DaaS_SS_ModbusTCP_Full_vx-y.zip`

App Starter Solution

All the libraries (dll) necessary to run the Application are provided.

For this Starter Solution, one package must be installed:

- .Net Framework (v4.7.1)

The offline installer of the .Net Framework v4.7.1 is provided in the Application zip file.

Full Starter Solution

The provided package includes:

- the source code of the application,
- the associated visual studio project,
- the necessary libraries (dll),
- the executable of program.

The source code is customizable and must be compiled to be used.

For this Starter Solution, three packages must be installed:

- Microsoft Azure Servicebus (v3.1.0)
- .Net Framework (v4.7.1)
- EasyModbusTCP (v5.5)

Pre-Run Setup

Overview

Before running this Starter Solution:

- If using the App Starter Solution, the Customer should launch the installer `NDP471-KB4033342-x86-x64-AllOS-ENU.exe`
- The Customer should proceed with the installation for the package content.
- The Customer should update the two following configuration files:
  - `CONNECTION_STRING_CFG.txt` is used to store the connection string of the Customer DaaS Queue. Copy / paste your connection string into this file.
  - Then, in the `DEVICE_ID_TABLE_CFG.txt` file, the Customer is expected to put the list of all his XIOT11SERMRCL transmitters that are expected to

send frames. This table is further used to map transmitters' data into the Modbus TCP Server registers.

NOTE: The format expected in this file is hexadecimal format in upper case (`00AABBCC`) with one Transmitter ID per line.

Once done, the Customer can run the Application by executing the executable file.

NOTE: If using the App Starter Solution, the executable file named `XIOTDaaSStarterSolution_ModbusTCP.exe` is located in the folder `Application\Release`.

Behavior of the Starter Solution

Overview

The `XIOTDaaSStarterSolution_ModBusTCP` will do the following:

Step	Action
1	Connect to an Azure Service Bus Queue (the DaaS Customer Queue) using the Connection String provided in the configuration file.
2	Set up a local Modbus TCP Server
3	Consume messages from the DaaS Customer Queue.
4	Display the received messages in the Console.
5	Store the received messages data in several Modbus Registers.
6	Clear the messages from the queue.
7	Close the connection to the DaaS Customer Queue and stop the Modbus TCP Server if the Customer presses any key twice.

This Starter Solution is expected to be used as following:

Step	Action
1	Check that Data-as-a-Service is activated on Cloud Connected Sensor Web platform.
2	Set up the Starter Solution by updating connection information for the DaaS Customer Queue and Device ID Table (configuration files).
3	Run the Starter Solution.
4	Trigger message sending by switching one of the two dry contacts connected to the XIOT11SERMRCL transmitter.
5	The messages sent by the transmitter are displayed in a console and the data are available in several registers of the Modbus TCP Server.
6	The Customer can then initiate a Modbus TCP communication with his SCADA / monitoring system to read the data in the corresponding registers.

Code Customization

The Starter Solution for Modbus TCP is provided as a basic example. The code must be adapted to the use case / environment.

For example, one thing that may be interesting for the Customer, is to customize the register mapping according to his needs.

NOTE: This Starter Solution has been implemented to parse only Measurement frames and Keep-Alive frames retrieved from the DaaS Customer Queue. To parse other frames, the existing code must be modified or another methodology must be implemented.

Modbus Server Configuration and Register Mapping

The Modbus Server initiated by the program starts listening on the active IPv4 address, on the port 502. The Modbus Server has the default unit ID 1. Unit ID and

Port can be tuned in the source code, however, no IPv4 address selection is possible at the moment.

The data sent by a transmitter are saved in the Modbus Server registers. For this purpose, application use only Holding Registers. The Modbus Server has a capacity of 65535 Holding registers.

One transmitter is represented using 50 registers. Thus theoretically, this application can handle more than 1200 transmitters.

If needed, the Customer can adapt the source code, to select and store only the data that are relevant for him.

For each new transmitter, the registers in which the data of this transmitter can be found are shifted by 50. For example, if the Customer has declared in the file `DEVICE_ID_TABLE_CFG.txt` two device ID (=transmitter ID) 00AABBCC and 00DDEEFF. The data belonging to the device 00AABBCC can be found in the registers 40001 to 40050. The data belonging to the device 00DDEEFF can be found in the registers 40051 to 40100.

The data are stored in the 50 registers as follows:

Address offset	Data	Format	Corresponding JSON attribute
1	Transmitter ID	ASCII representation of the 2 first character of the transmitter ID (**00**AABBCC).	Device
2	Transmitter ID	ASCII representation of the 2-next character of the transmitter ID (00**AA**BBCC).	Device
3	Transmitter ID	ASCII representation of the 2-next character of the transmitter ID (00AA**BB**CC).	Device
4	Transmitter ID	ASCII representation of the 2-last character of the transmitter ID (00AABB**CC**).	Device
5	Timestamp	16 bits representation of the timestamp year.	Time
6	Timestamp	16 bits representation of the timestamp month.	Time
7	Timestamp	16 bits representation of the timestamp day.	Time
8	Timestamp	16 bits representation of the timestamp hours.	Time
9	Timestamp	16 bits representation of the timestamp minutes.	Time
10	Timestamp	16 bits representation of the timestamp seconds.	Time
11	Average SNR	16 bits representation of the transmitter Average SNR. Value is truncated and represented as an unsigned value. Not provided anymore by Sigfox (always " null").	avgSnr
12	SNR	16 bits representation of the transmitter SNR. Value is truncated and represented as an unsigned value.	snr
13	Station ID	ASCII representation of the 2 first character of the Sigfox base station that received the Sigfox message (**AA**BB).	Station
14	Station ID	ASCII representation of the 2 first character of the Sigfox base station that received the Sigfox message (AA**BB**).	Station
15	Latitude	16 bits representation of the Sigfox base station latitude. Value is truncated and represented as a signed value. Not provided anymore by Sigfox (always " null").	lat
16	Longitude	16 bits representation of the Sigfox base station longitude. Value is truncated and represented as a signed value. Not provided anymore by Sigfox (always " null").	lng
17	RSSI	16 bits representation of the transmitter RSSI. Value is truncated and represented as a signed value.	rssi
18	Sequence Number	See JSON frame description.	seqNumber
19	Record Type	See JSON frame description.	RecordType
20	Frame Counter 1	See JSON frame description.	FrameCnt1

Address offset	Data	Format	Corresponding JSON attribute
21	HW Error	See JSON frame description.	HWError
22	Low Bat Error	See JSON frame description.	LowBatError
23	Sensor 1 Closed Counter	See JSON frame description.	S1ClosedCnt
24	Sensor 2 Closed Counter	See JSON frame description.	S2ClosedCnt
25	Sensor 2 Previous State	See JSON frame description.	S2PreviousState
26	Sensor 2 Current State	See JSON frame description.	S2State
27	Sensor 1 Previous State	See JSON frame description.	S1PreviousState
28	Sensor 1 Current State	See JSON frame description.	S1State
29	Sensor 1 Open Counter	See JSON frame description.	S1OpenCnt
30	Sensor 2 Open Counter	See JSON frame description.	S2OpenCnt
31	Transmitter Latitude MSB	32 bits floating representation of the transmitter latitude. In this register, only the first 16 bits (MSB) are present..	lat
32	Transmitter Latitude LSB	32 bits floating representation of the transmitter latitude. In this register, only the second 16 bits (LSB) are present..	lat
33	Transmitter Longitude MSB	16 bits representation of the Sigfox base station longitude. In this register, only the first 16 bits (MSB) are present.	lng
34	Transmitter Longitude LSB	16 bits representation of the Sigfox base station longitude. In this register, only the second 16 bits (LSB) are present..	lng
35	Sensor 3 Previous State	See JSON frame description.	S3PreviousState
36	Sensor 3 Current State	See JSON frame description.	S3State
37	Sensor 4 Previous State	See JSON frame description.	S4PreviousState
38	Sensor 4 Current State	See JSON frame description.	S4State
39	RESERVED	RESERVED FOR FUTURE USE	N/A
…	…	…	…
49	RESERVED	RESERVED FOR FUTURE USE	N/A
50	Not Alive Flag	Value set to 1 when no frames (Keep Alive or Measurement) have been received since the last 25 hours for this device.	N/A

The information provided in this document and in the source code could change in the future. It is recommended that the Customer consider this example as a “prototype code” or as a “proof of concept”.

Data management:

- The registers of the server are accessible from Modbus/TCP clients in read-only mode
- The Modbus/TCP server does not store the history of the data
- The values of all the registers are reseted after each start of the Modbus/TCP server

Appendices

What's in This Part

Publish / Subscribe Messaging Principle and ETL Processes

What's in This Chapter

ETL Processes

ETL Processes Overview

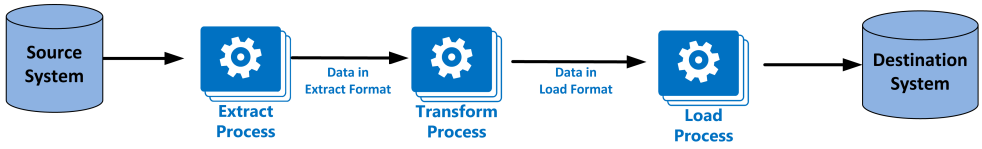
Overview

This section provides common concepts and terminology required for the Cloud Connected Sensor Data-as-a-Service solution.

The following figure presents the basic ETL concepts including processes, systems and data flow (indicated by arrows):

- The **Extract Process** extracts data from the Source System and formats it into a structured Extract Format.
- The **Transform Process** transforms the Extract Format data into a Load Format.
- The **Load Process** loads the Load Format data into the Destination System.

ETL Logical Components



Independently of how these processes are implemented, Extract, Transform and Load are common to all integration scenarios. Also, ETL always occurs in this order: first Extract, then Transform, then Load.

Extract Process

Overview

The role of the Extract Process (also called Consume Process) is to extract data from the Source System and output the data into the Extract Format so that it is consumable by the Transform Process.

The Source System can be any type of data storage including these common ones:

- Database (Relational Database, NoSQL DB, Graph DB, etc.)
- SCADA Historian
- File System (Logs, Analytic outputs etc.)
- Messaging System

Transform Process

Overview

It may be required to transform data from Extract Format to Load Format before data can be loaded into a Destination System. This is the role of the Transform Process.

The Transform Process is often implemented as part of the Load Process and not seen as separate, but it is an independent process regardless of how it is coded or deployed. After the data is transformed into a Load Format understood by the Destination System (database prepared statement or file operations, etc.) it can be loaded.

The Extract Format is required for the development scalability to support diverse Destination Systems. Because the Extract Format is Destination System agnostic, the Extract Process is decoupled from all Destination System specifics. The result of this decoupling is a very simple Extract Process and an extensible integration

architecture, meaning a Destination System agnostic Extract Format can be loaded into any Destination System. The Destination System owner or system integrator is responsible for the creation of specialized Transform and Load Processes.

As well as transforming from Extract to Load Formats, the Transform Process must include data validation because invalid (yet correctly formatted) data can cause corruption and data loss in Destination Systems.

Some topics of data validation are:

- Data-type
- Range Checks
- Constraints
- Referential Integrity (e.g. Relational Databases)
- Structure
- Nulls Allowed
- Duplication

The ultimate result of Data Transform is a set of data which is prepared to be loaded into the Destination System.

Load Process

Overview

After transformation, data is read by the Load Data Process and loaded into the Destination System. The Load Data Process executes basic database operations using pre-validated, pre-formatted data with minimal processing.

The Load Data Process must be simple and robust. If the data load was not successfully finished, the assumption should be (and logging should demonstrate) that it was due to an external reason such as an incorrect behaviour of the Transform Process, a network or connectivity interruption or an issue with the Destination System (e.g. full disk).

Process activity logging is often required by Transform and Load Processes for auditing purposes.

Publish/Subscribe Messaging

Overview

Publish/Subscribe Messaging (or Messaging) is a Push integration strategy in which a data Producer pre-emptively pushes data messages to a Consumer. Messaging presents many integration advantages, very few disadvantages and is robust, proven and secure across company-to-company boundaries.

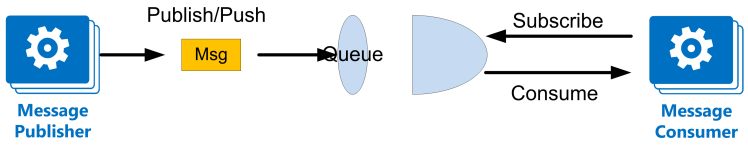
Basic Concept

Overview

The basic concept of messaging is:

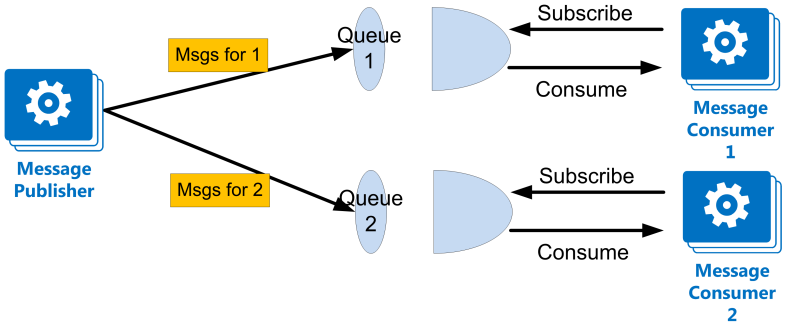
- Publisher hosts a data storage and communications mechanism. This will be called a Queue.
- Consumer Subscribes to this Queue
- Data Publisher Pushes data onto this Queue in the form of a Message
- Consumer receives the Message and processes the Data

Publish/Subscribe Messaging:



In public scenarios, the Queue is a publicly available network resource, such as a news or weather feed, available to any Consumer which wishes to receive messages from it. In a Data-as-a-Service integration scenario the Queue would not be publicly available and only DaaS Customers would be able to Subscribe to the Queue. In addition, a DaaS Customer only receives those messages which belong to it; it cannot receive messages belonging to any other Customer.

Publish/Subscribe with Multiple Consumers:



Each Consumer (or DaaS Customer) subscribes to a logical queue dedicated to it. It receives only messages send to its queue. A Consumer has no visibility to other Consumer messaging queues, has no credentials for them, and cannot subscribe to them.

Note on Cybersecurity

Overview

The Message Consumer uses outbound network connections to the Message Queue and does not need to expose a public network endpoint. Cybersecurity concerns for Messaging-based integration are therefore limited, especially for Customers consuming messages on-premise.

In Messaging architectures, the Message Producer (Schneider Electric / Telemecanique Sensors in Cloud Connected Sensor case) takes on the security risk: hosting, securing, managing, and exposing the network resource (the Queue).

To receive its messages, the Message Consumer must initiate a session to the Queue and query it. Rather than having the Queue connected to the Message Consumer via a public network endpoint, the Message Consumer initiates an outbound connection from the Customer network to the Queue network endpoint. While there are some differences, this is analogous to a web client connecting to a web server to request web site content and so presents limited network security risk for the Customer.

It is also possible that the Message Consumer connects to the Queue for a short period before disconnecting and reconnecting. This further reduces cyberattack risk because the Message Consumer use a non-determinant sequence of volatile outbound ports.

Information Concerning Technologies Used

What's in This Chapter

Microsoft Azure Service Bus

Overview

The DaaS Customer Queue is an Azure Service Bus Queue. Information about this technology is available from the web site of Microsoft.

Here is a link to the Azure Service Bus Messaging documentation.

https://docs.microsoft.com/en-us/azure/service-bus-messaging/

On this page all the key information concerning Azure Service Bus can be found:

- What is Azure Service Bus?
- How it works?
- Tutorials and example code to interact with Azure Service Bus
- Any other key information

Microsoft Visual Studio

Overview

The Starter Solutions are provided as Microsoft Visual Studio .NET Core console application. Information about this Integrated Development Environment is available from the web site of Microsoft.

Here are two links to the Visual Studio documentation:

- Visual Studio General Information: https://visualstudio.microsoft.com/
- Visual Studio Documentation: https://docs.microsoft.com/en-us/visualstudio

Glossary

D

Destination System:

the system owned by Destination Partner into which the Load Formatted data is Loaded by the Load Process.

E

ETL Processes:

the three processes implemented by the message consumer:

- Extract process
- Transform process
- Load process

Extract Format:

the format data is written to by the Extract Process for use by the Transform Process.

Extract Process:

the process which extracts data from the Source System and writes to the Extract Format.

I

Integration Partners:

the partners involved in an integration solution. Partners could be called Source Partner and Destination Partner. In this case, Cloud Connected Sensor is the Source Partner and the DaaS Customer is the Destination Partner.

J

JSON:

JavaScript Object Notation is a terse, readable, structured data format used for Device-to-Cloud and Cloud-to-Device messaging. Many stream processing applications are built to natively consume JSON structures.

L

Load Format:

the format data is written to by the Transform Process for use by the Load Process.

Load Process:

the process which loads data Load Format into the Destination System.

S

Source System:

system owned by Source Partner which contains the original data which is extracted by the Extract Process.

T

Transform Process:

the process which transforms data in Extract Format into data in Load Format.

Schneider Electric
35 rue Joseph Monier
92500 Rueil Malmaison
France

+ 33 (0) 1 41 29 70 00

www.tesensors.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.

EIO0000003797.01