How to connect BMS or other data streams to iSCAN

20th October 2020

Strictly Confidential

Data to iSCAN

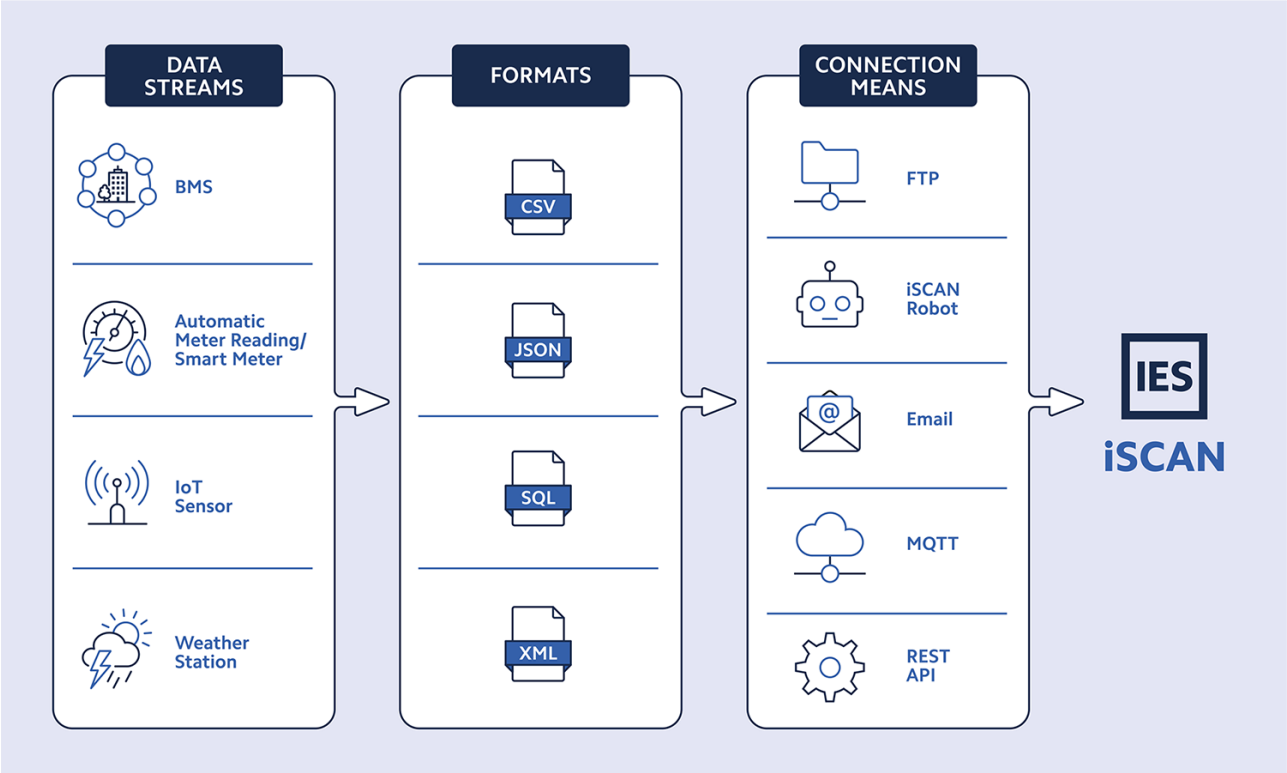
To collect data from a BMS network and import it to iSCAN the following options are available:

- Set the system to periodically export histories of points to a local folder in the client's network (CSV, XML, Json or oBIX). Install a service (iSCAN-robot) on the network that automatically gathers the file at regular intervals and uploads the data to iSCAN.
- Set the system to export periodically histories of points to an FTP server. The vendor will create a program to retrieve data periodically from the FTP directory and upload it to iSCAN.
- Set the system to send exported files to an iSCAN generated email address. This will allow automated data upload to iSCAN
- If the system can function as an MQTT client, data can be uploaded in iSCAN using the MQTT protocol.

For any of the import methods, to be able to connect to iSCAN or send emails directly to iSCAN by some automated means, an internet connection would be required.

Manual data imports are also possible if the data is collected from the site and transferred to a machine that can communicate with iSCAN. While no internet connection is required at the source of the data, this approach will require physical access to the machine where the data is being collected. Typically, manual uploads would fall into the CSV, XML, obix and Json data formats.

The diagram below illustrates the different data sources, file formats and connection means to iSCAN.



iSCAN robot

The iSCAN robot is a software that can be obtained from a building on the iSCAN website. When downloaded, installed on a computer and configured, the iSCAN robot can collect data from a computer or location on the same network as the machine on which it is installed and transmit it to the iSCAN website. This has several advantages:

- It runs automatically and regularly keeps data on iSCAN up to date with the latest on your local machine or network location.
- It runs as a windows service, so there is no application for a user to accidentally close, and if power is lost for any reason, it will start running as normal once the computer is turned back on without any user intervention.
- After the initial configuration of permissions for the iSCAN robot, all further configuration is done via the iSCAN website and changes are automatically sent to the website.

In case iSCAN-robot service will be required, it will be installed on a machine which has access to the resources that will be pushed to iSCAN platform. The robot will use a unique key (token) to request the pushed data source for the building. The robot can be downloaded from iSCAN website and it will be installed with an installer generated using the unique key. The installed robot service will put its data in a folder under %programData%\IESRobot.

The robot service installs executable files under %ProgramFiles(x86)%\ScanRobot.

The robot service stores data and configuration in %ProgramData%\ScanRobot.

Data files waiting to be uploaded to iSCAN are stored in the %ProgramData%\ScanRobot\data sub-folder and an activity log is written in the %ProgramData%\ScanRobot\log sub-folder.

The robot service configuration is via an XML file %ProgramData%\ScanRobot\config.xml.

Resetting the key will prevent any installed robots from pushing data until a new key will be provided. This should be done if the key has been made public, as anyone with the key could access to data source settings stored on the iSCAN website for the building.

Processing and performance requirements

Many automated means of data collection rely upon collecting and uploading data at regular intervals. The files should be set to uploaded at the same frequency at which they are generated and this interval can vary by project. Common frequencies for data collection and uploading are:

- Daily
- Hourly
- Half-Hourly
- Every 10 minutes

Supported File Structures

iSCAN supports a number of flexible file formats. Common formats generated by BMS software and supported by iSCAN include the following:

- CSV
- XML, oBIX, SQL, MQTT

- SENML Json

The following Appendix describes each of the formats adding some context to them.

Appendix- Examples of formats and protocols

.csv file format

A sample of the structure of suitable csv files are shown below:

1. Row per timestamp, column per data point

Timestamp	Data point 1	Data point 2	Data point 3	Data point n
23/06/2020 10:00	10	17	9	11
23/06/2020 10:30	0	14	16	6
23/06/2020 11:00	11	14	12	4
23/06/2020 11:30	20	12	34	26
23/06/2020 12:00	23	24	22	12
23/06/2020 12:30	24	43	21	23

2. Row per point. Each row will have a different point

Timestamp	Data points	Value	Units
23/06/2020 10:00	Data point 1	10	C
23/06/2020 10:00	Data point 2	17	C
23/06/2020 10:00	Data point 3	9	C
23/06/2020 10:00	Data point n	11	C
23/06/2020 10:30	Data point 1	0	C
23/06/2020 10:30	Data point 2	14	C
23/06/2020 10:30	Data point 3	16	C
23/06/2020 10:30	Data point n	6	C

3. Rectangular. Each row represents a different day.

123456	DataPoint1	12/05/2020	1	2	....	4	4	kWh
123456	DataPoint1	13/05/2020	2	0	0	0	5	kWh
123456	DataPoint1	14/05/2020	8	1	0	0	0	kWh
123456	DataPoint1	15/05/2020	5	4	2	5	4	kWh
123456	DataPoint1	16/05/2020	0	0	0	0	0	kWh
123456	DataPoint1	17/05/2020	0	0	0	0	0	kWh
123456	DataPoint1	18/05/2020	5	0	0	4	0	kWh
123456	DataPoint1	19/05/2020	0	4	0	0	0	kWh

This format presents a different structure. Each row represents a different day and the number of columns suggest the frequency (i.e. 30 mins frequency data will have 48 columns of values, 60 mins frequency 24 columns of values and so on).

For all three options, timestamp should be increasing i.e. 23/06/2020 10:00, 23/06/2020 10:30, 23/06/2020 11:00. If timestamp is decreasing or random order, then import will fail.

.xml file format

A sample of the structure of a suitable xml file is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<scan-data>
  <channel id="DataPoint1">
    <p date="2020-09-29T00:00:00.0000000ZZ" val="1" />
    <p date="2020-09-29T00:10:00.0000000ZZ" val="2" />
    <p date="2020-09-29T00:20:00.0000000ZZ" val="3" />
    <p date="2020-09-29T00:30:00.0000000ZZ" val="4" />
    <p date="2020-09-29T00:40:00.0000000ZZ" val="5" />
    <p date="2020-09-29T00:50:00.0000000ZZ" val="6" />
    <p date="2020-09-29T01:00:00.0000000ZZ" val="7" />
  </channel>
  <channel id="DataPoint2">
    <p date="2020-09-29T00:00:00.0000000ZZ" val="1" />
    <p date="2020-09-29T00:10:00.0000000ZZ" val="2" />
    <p date="2020-09-29T00:20:00.0000000ZZ" val="3" />
    <p date="2020-09-29T00:30:00.0000000ZZ" val="4" />
    <p date="2020-09-29T00:40:00.0000000ZZ" val="5" />
    <p date="2020-09-29T00:50:00.0000000ZZ" val="6" />
    <p date="2020-09-29T01:00:00.0000000ZZ" val="7" />
  </channel>
  <channel id="DataPoint3">
    <p date="2020-09-29T00:00:00.0000000ZZ" val="1" />
    <p date="2020-09-29T00:10:00.0000000ZZ" val="2" />
    <p date="2020-09-29T00:20:00.0000000ZZ" val="3" />
    <p date="2020-09-29T00:30:00.0000000ZZ" val="4" />
    <p date="2020-09-29T00:40:00.0000000ZZ" val="5" />
    <p date="2020-09-29T00:50:00.0000000ZZ" val="6" />
    <p date="2020-09-29T01:00:00.0000000ZZ" val="7" />
  </channel>
  <channel id="DataPointn">
    <p date="2020-09-29T00:00:00.0000000ZZ" val="1" />
    <p date="2020-09-29T00:10:00.0000000ZZ" val="2" />
    <p date="2020-09-29T00:20:00.0000000ZZ" val="3" />
    <p date="2020-09-29T00:30:00.0000000ZZ" val="4" />
    <p date="2020-09-29T00:40:00.0000000ZZ" val="5" />
    <p date="2020-09-29T00:50:00.0000000ZZ" val="6" />
    <p date="2020-09-29T01:00:00.0000000ZZ" val="7" />
  </channel>
</scan-data>
```

IES can provide an .xslt file to parse data from a .xml file and facilitate the import.

.obix file format

A sample of the structure of a suitable .obix file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<obj href="http://ObjectName*" xmlns="http://obix.org/ns/schema/1.0">
 <obj>
  <abstime name="timestamp" val="2020-10-21T00:00:00.000+01:00" display="21-Oct-20 4:45 AM BST" tz="Europe/London"/>
  <str name="DataPoint1" val="0" display="{ }"/>
  <str name="Status" val="0" display="{ok}">
  </str>
  <real name="value" val="0.0" display="0.00"/>
 </obj>
 <obj>
  <abstime name="timestamp" val="2020-10-21T00:10:00.000+01:00" display="21-Oct-20 4:45 AM BST" tz="Europe/London"/>
  <str name="DataPoint1" val="0" display="{ }"/>
  <str name="Status" val="0" display="{ok}">
  </str>
  <real name="value" val="0.0" display="0.00"/>
 </obj>
 <obj>
  <abstime name="timestamp" val="2020-10-21T00:20:00.000+01:00" display="21-Oct-20 4:45 AM BST" tz="Europe/London"/>
  <str name="DataPoint1" val="0" display="{ }"/>
  <str name="Status" val="0" display="{ok}">
  </str>
  <real name="value" val="0.0" display="0.00"/>
 </obj>
 <obj>
  <abstime name="timestamp" val="2020-10-21T00:30:00.000+01:00" display="21-Oct-20 4:45 AM BST" tz="Europe/London"/>
  <str name="DataPoint1" val="0" display="{ }"/>
  <str name="Status" val="0" display="{ok}">
  </str>
  <real name="value" val="0.0" display="0.00"/>
 </obj>
</obj>
```

The object is the root of the “tree” that structures the .obix file. Its href attribute identifies the URI for the OBIX document.

In this example, we see an object which has two children in the tree, DataPoint1 and Status.

The Status will just suggest the status of the object. T

he <real> object stores the value of the points - in this case it stores the value of DataPoint1.

The <abstime> row gives a timestamp to the values.

.JSON file format

A sample of the structure of a suitable .json file is shown below.


```
[{"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.7, "bt": 1587250965.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.6, "bt": 1587251266.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.5, "bt": 1587251568.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.4, "bt": 1587251869.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.3, "bt": 1587252171.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.2, "bt": 1587252473.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21.1, "bt": 1587252774.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 21, "bt": 1587253076.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 20.9, "bt": 1587253377.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 20.8, "bt": 1587253679.0},
 {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0},
 {"n": "DataPoint2", "v": 20.7, "bt": 1587253981.0}]
```


In the object {"n": "DataPoint1", "v": 28.6, "bt": 1587250965.0}:

- n = the unique ID of the sensor/meter
- v = the record value
- bt = the unix timestamp of the record

SQL database

Using the iSCAN robot, it is possible to extract data from an SQL database and upload data into iSCAN as .xml file.

To do so, the iSCAN robot config file will need to enable SQL connection adding the tag <AllowSQL>True</AllowSQL> so the robot will be allowed to communicate with a database.

Only postgresql and SQL server are currently supported.

For robot configuration in iSCAN , the url needs to be of the following form:

<postgresql or sqlserver>://<address of database, either URL or IP address>:<port optional>/<name of database>

The robot requires the connection details for a user of sufficient privileges to issue the queries.

MQTT protocol

Using the MQTT protocol, IES will connect as a client to the server and query it to receive some data into IES servers.

IES will need access to the MQTT broker to retrieve data and it will parse it to upload queried data into iSCAN as .xml or .json files.