Title:	**IPA Detailspecification Loris Isenegger**
Subject:	**Detailspecification**
Project:	**00001, Cloud Applications**

This document specifies the requirements for the IPA of Loris Isenegger.

Key Words: IPA, Database, Preprocessing

Document Storage:	Local
Document Category:	ProjectRecord
Revision:	0.11
Revision Date:	2019-04-11
Document Status:	Working
Author:	Young Ban
Department:	BT CPS R&D ZG CS SAP
Responsible:	banyoung@siemens.com
Company:	Siemens Schweiz AG, Building Technologies Division Control Products & Systems
Classification:	Restricted
Based on Template:	Workbook_Standard; 4; 2014-11-05; Donat Hutter, 3531

Revision History

Rev	Date	Author	Remarks
0.11	11-Apr-2019	Michael Speckien, 5556	Status = **Working** - Testing more details
0.1	22-Feb-2019	Young Ban	Status = **Working** - Initial Creation

Table of Contents

1. Introduction

1.1 Purpose of the document

This document specifies the task for the IPA of Loris Isenegger.

1.2 Scope, Field of application

In current times, data is abundant and although this can be valuable, it often needs some preprocessing first. In the field of building automation, fault detection plays a significant role in saving costs.
Although there is an existing application which analyzes measured sensor/device data to detect faults, extracting this data from the database is not yet implemented.
The general goal is to implement an efficient way to read and structure data from the database on a daily basis so that it can be used by this application. In this specific IPA the looked at example is the FDD application for heat pumps for preventative maintenance.

This document provides a detailed task description of what should be worked on during the entire duration of the IPA. Since the focus is on the FDD application for heat pumps, a specific set of datapoints related to heat pumps, a time interval, a sampling time and a location are provided in form of an .csv file.
Although a specific use case for heat pumps is given, the implementation of this project should be done in such a way that it is able to handle jobs by other application if they are given in the same format as the one mentioned above. The tool should be able to handle multiple jobs so that in the optimal case there is only one instance of this tool running.

1.3 Glossary

Term	Description
FDD	Fault detection & diagnosis
.csv	File format where the values are assumed to be separated by a delimiter (generally: comma per default; in the scope of this IPA: space)
InfluxDB	Name of the database type used in this project

2. Current Situation

2.1 Overview database

The device and sensor data are stored in a time series database called **InfluxDB**. The database is sub-divided into measurements. Each line in the measurement has a timestamp, one/more tags and fields.
In the Siemens environment the convention is that each line entry of a measurements always contains following tags: a **device-obj-instance**, an **object-name**, an **object-type** and an **object-instance**, which together uniquely define the datapoint. The fields, which consist of the **value** and their respective **unit**, describe what is actually measured.

2.2 Example use case: Fault Detection & Diagnosis

The idea in this use case is to analyze the gathered data from the various sensors and devices in our plant to predict problems with the installed hardware and software. By reading and providing an output file in the right format, this project should implement the connection point between the database and the mentioned application. The analysis is done daily based on the data of the previous day.

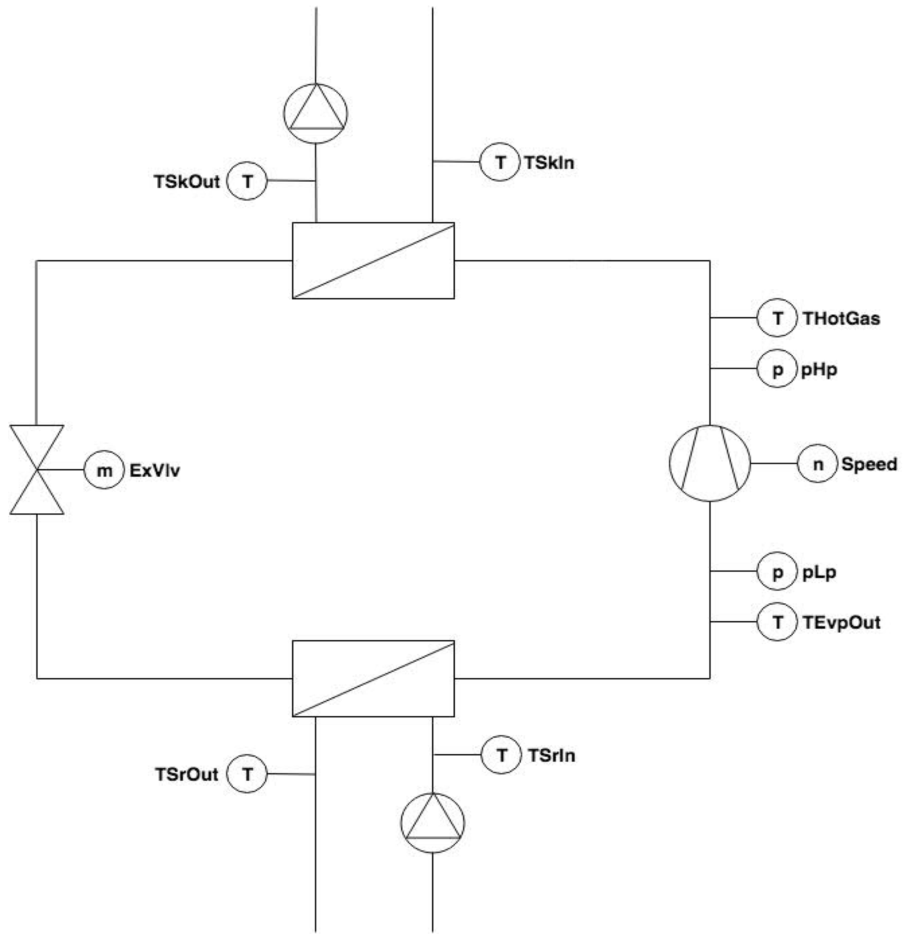


Figure 1: Example of a measurement point plan

3. Goal of this project

As previously mentioned, a tool should be created to make the gathered data available to other applications. This tool should mainly consist of a build part and a service part. By manually providing an appropriate config and mapping file to the build part and executing it, the administrator of an application can register a job. The tool should then extract the data specified in the files and return it in a standard format at the specified location. To match the needed time resolution of the data an interpolation step might need to be performed. The implementation should be able to handle the jobs of multiple applications and provide new output files at the end of each day. This also means that if a job is not manually deleted/unregistered, the tool will continue to provide output files for each registered job, overwriting the old ones, every day.

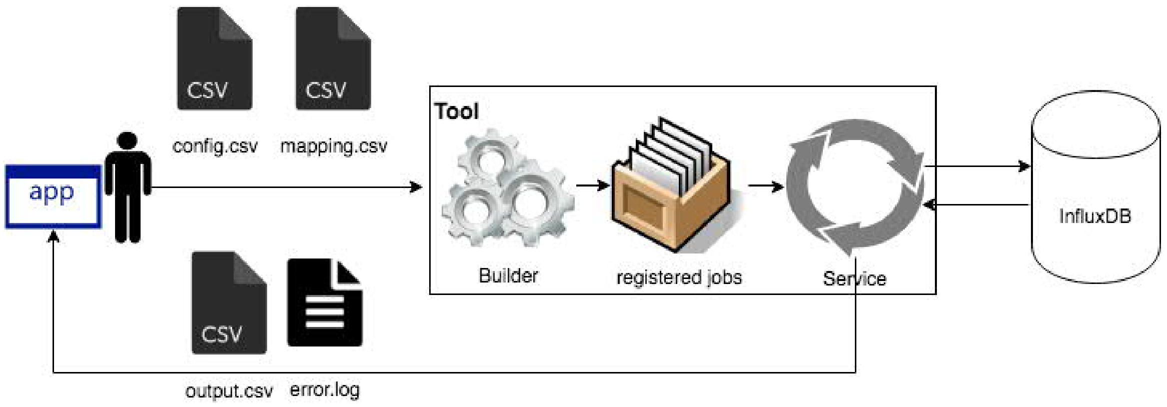


Figure 2: Workflow

Example of a workflow:

- **Day 1**: **Application A's administrator** provides config + mapping file to the tool and executes the build part to register a new job
- **Day 1**: Tool uses config and mapping file to construct appropriate query/queries for **Application A**
- **End of Day 1**: Tool stores queried data of **Day 1** in the correct format at the location specified in the config file for **Application A**
- **Day 2**: *Application A may/may not use the output file provided by the tool a day before*
- **End of Day 2**: Tool stores queried data of **Day 2** in the correct format at the location specified in the config file for **Application A** overwriting the data from the previous day

- **Day 3**: *Application A may/may not use the output file provided by the tool a day before*
- **Day 3**: **Application B's administrator** provides config + mapping file to the tool and executes the build part to register a new job
- **Day 3**: Tool uses config and mapping file to construct appropriate query/queries for **Application B**
- **End of Day 3**: Tool stores queried data of **Day 3** in the correct format at the location specified in the **respective** config file for **both Application A and B** overwriting the data from the previous day

- **Day 4**: *Application A and B may/may not use their respective output file provided by the* tool

- …

4. Detailed requirements

4.1 Structure and dataflow

The used programming language should be **Python** (3.6) and the implemented project should be called *InfluxDExTool* (InfluxDataExtractionTool). As previously mentioned, it should mainly consist of two parts. The build part should be called *JobBuilder*, which is responsible for building new jobs and storing those for later use by the regularly running service part, which should be called *DExService*.
To register a job the administrator of an application enters the **paths** of the files and the **job name** into a tool internal **register file**. By executing the *JobBuilder*, this file will be fully read, and the list of registered jobs will be updated. This means that to unregister a job the related entry in the register file must be deleted and the *JobBuilder* has to be executed again. Each job registered consists of a **job name**, an **output location**, the related **queries** constructed by the *JobBuilder*.
At the end of every day *DExService* will access the list of registered jobs, use the stored queries to extract the respective data and create a **folder** at the specified location named after the **name of the job**. This folder contains an **output file** in the .csv format (delimited by spaces), where the extracted data is stored and a **log file**, which documents errors **during the execution** of *DExService* for this job. The output file should be **named "JobName"_ output.csv** and the log file should be named **"JobName"_error.log**, where **"JobName"** corresponds to the **name of the job entered into the register file**.

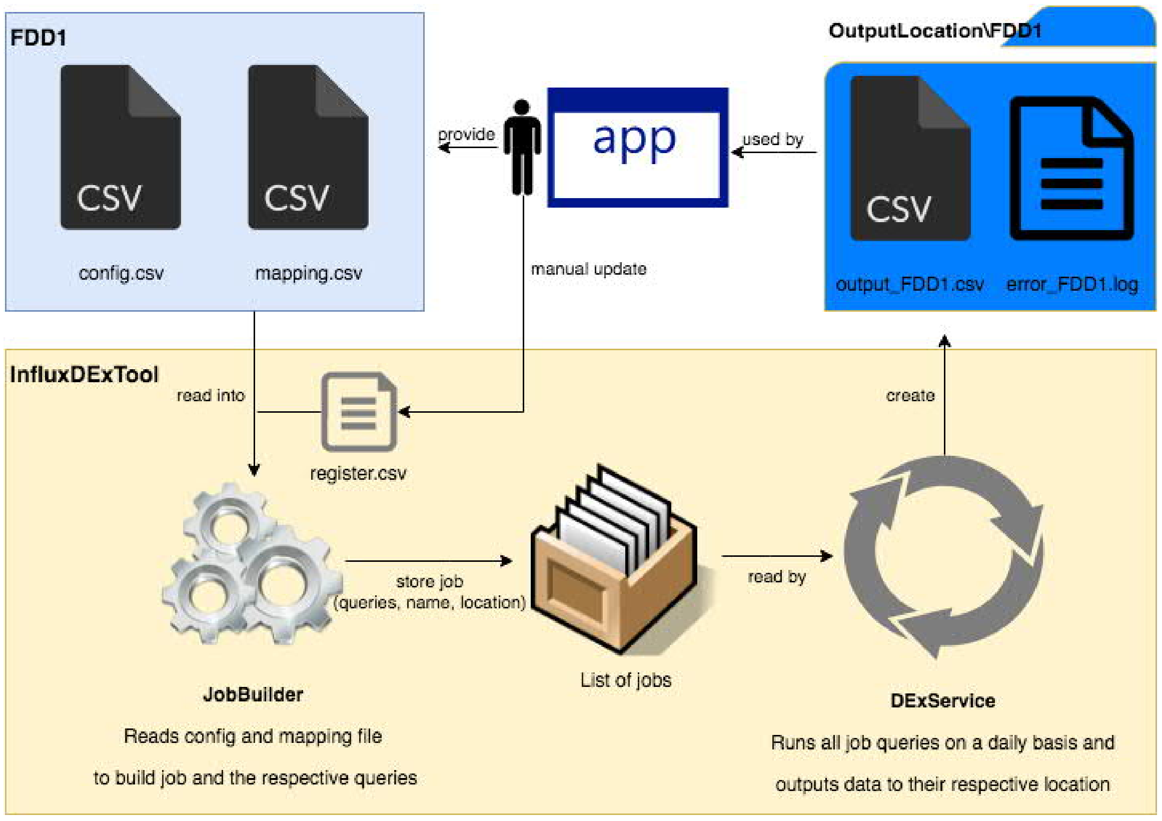


Figure 3: Dataflow

4.2 Configuration file

The relevant information about time interval, sampling time and data points is given by each application as a configuration file in the .csv format (delimited by the space character).

- The time interval describes between which **start time** and **end time** the timestamps of the extracted data should be. Those times are given in the format **HH:MM:SS** and always lie between 00:00:00 and 24:00:00.
- The **sampling time** in **seconds** defines the time difference between each line.
- The **output location** is provided as a string and is assumed to be always accessible **inside the siemens intranet**.
- A **list of the requested datapoints**: **Col** describes in **which column** the values of the respective datapoint should be stored in the output file; **Interpolation** describes how missing data should be constructed (**Note:** Only **linear** and **previous** are the only allowed strings)

Config				
StartTime	08:00:00			
EndTime	14:00:00			
SamplingTime	10			
OutputLocation	\\ch021012\SIM_MOD\IPA_Loris			
Datapoints				
Col	ID	Comment	Unit	Interpolation
1	TSrIn	PT100	[°C]	linear
2	TSrOut	PT100	[°C]	linear
3	TSkIn	PT100	[°C]	linear
4	TSkInTSkOut	PT100	[°C]	linear
5	pLp	PA-33X	[barg]	linear
6	pHp	PA-33X	[barg]	linear
7	THotGas	NTC	[°C]	linear
8	TEvpOut	NTC	[°C]	linear
9	ExVlv	Carel198	[%]	previous
10	Speed	compressor	[Rpm]	linear

Table 1: Example config file**Error! Not a valid link.**

4.3 Mapping

Note that the datapoints in our database are not directly identified by the list of names provided by the configuration file. As described in 2.1 each datapoint is uniquely identified by the combination of tags. The mapping between the names specified in the config file and these tags should be inferred from the mapping file provided as another .csv file (delimited by the space character) in addition to the config file.

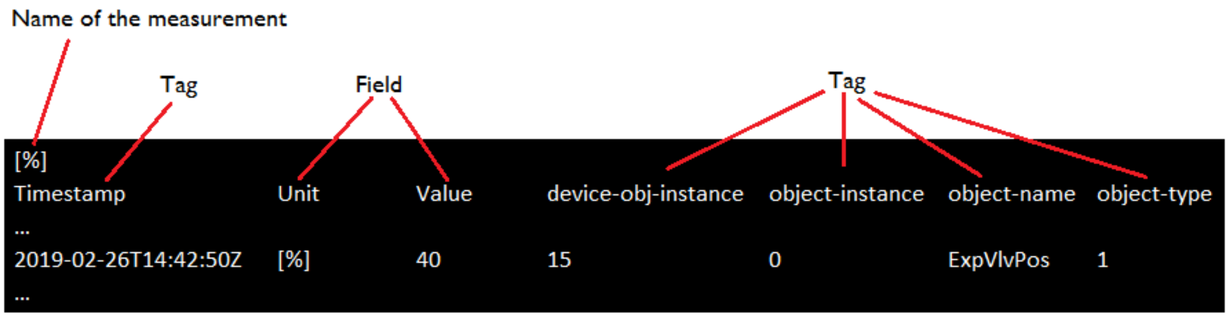
Error! Not a valid link.
Table 2: Example of a mapping file

4.4 Accessing the database

The type of database used is a time-series based database called InfluxDB. To query this database a language very similar to SQL can be used. There exist a few custom functions which are tailored to the time-series based nature. One task of this project is to read the necessary data efficiently making use of the structure of the database.

Although the tool will be run on the cloud, in the scope of this IPA the tool should only be run locally. Also, to guarantee access to the database during the whole duration of the IPA, a local version of InfluxDB will be set up and filled beforehand.
To access the database:

- The database is **hosted** at: **localhost** with **port**: **8086**
- The name of the **database** is **IPA_DB** and it consists of measurements each identified by the **unit** of the values.
- In each measurement the **tags** are**: device-obj-instance, object-name, object-type and object-instance**, which together uniquely define a datapoint
- The actual value of the datapoint is given in the **field**: Value
- The unit is given in the **field**: Unit



```
[%]
Timestamp              Unit   Value   device-obj-instance   object-instance   object-name   object-type
...
2019-02-26T14:42:50Z   [%]    40      15                    0                 ExpVlvPos     1
...
```

Figure 4: Example of a measurment in InfluxDB

4.5 Format of the output

The resulting data from our tool should formatted as a .csv file such that each column delimited by a space character represent a data point. Which column corresponds to which data point is given by the entries under **Col** in our config file (0 corresponds to the first column).

Timestamp	TSrIn	TSrOut	TSkIn	TSkOut	pLp	pHp	THotGas	TEvpOut	ExVlv	Speed
[TT.MM.JJJJ hh:mm:ss]	[°C]	[°C]	[°C]	[°C]	[barg]	[barg]	[°C]	[°C]	[%]	[Rpm]
26.02.2019 07:00:10	2.876	2.985	39.275	39.332	8.833	8.844	51.522	11.438	-0.1	-0.5
26.02.2019 07:00:20	2.957	3.009	39.038	38.916	8.849	8.86	50.666	11.787	-0.1	-0.5
26.02.2019 07:00:30	3.231	3.009	38.923	38.566	8.873	8.886	49.857	12.135	103.3	-0.5
…										

Table 3: Example of an output file

4.6 Error Handling

Since the tool consists of two parts which completely differ in tasks, error handling should be done separately.

The JobBuilder is required to be able to detect/handle at least following errors:

- Detect if **files** corresponding to the jobs entered in the **register file exist/are accessible**
- Detect if there are **duplicate entries** in the register file, which would lead to conflicts (both job name and location are the same)
- Detect if **StartTime** and **EndTime** have right format in the config file
- Detect if **SampleTime** is number
- Detect if **OutputPath**, where the output should be written to exists/is accessible
- Detect if there is **no mapping** in the mapping file for certain names in the config file

If any such error is detected the JobBuilder should continue execution but display a hint in the console and log error related hints in a log file. Note that a **successful execution should be also indicated**.

The DExService part is should:

- Detect if **database is unavailable**
- Detect if **database entries do not exist**
  -> **Fill** corresponding columns in the output file **with dummy values**
  -> **Log** this occurrence in the **error log** mentioned in **4.1**

The error log should be delivered **together with the output file**. Note that a **successful execution should be also indicated**.

4.7 Testing

4.7.1 Module Test

- Role: Executed by the implementer
- When: as soon as a module is implemented
- Type: White box test
- Test object: the module with its input and output interfaces
- Test cases: use cases and exceptions: is the code correctly executed during debugging
- Test coverage: all major parts of the code shall be covered
- Documentation: For each module a list of test cases (one line per test case)

Remark: usually we do not include module tests into the project documentation, but for this IPA the module test shall be documented.

4.7.2 Product test

- Role: Executed by the product tester

Remark: usually the product test is not executed by the implementer, but for this IPA the product test shall be done by the implementer

- When: after implementation and integration of all modules is finished, that means when the product is ready for use.
- Type: black box test
- Test object: complete product
- Test cases: Deciding how many tests and what tests are necessary is part of the IPA.
- Documentation. For each test at least the following points should be documented:

- Description of the test and what it is testing
- What is the predicted result?
- What is the actual result?
- If the actual result does not match with the prediction: Why?

4.8 Documentation

Since the format of the config file and the mapping file is given, the user documentation on the application side (how the config file should look like, …) is **not** a required part of the IPA.

The following documentation **is required**:

- Developer documentation describing the functions used and the relations between each other (Nassi-Shneiderman-Diagram)
- Documentation of the libraries, data structures and files used
- User documentation of the tool (what files are needed, how to fill in the setting file, how to execute the scripts, description of the logfiles, …)

The goal of the documentation is that a future computer science intern can debug and extend the tool with minimal effort.

4.9 Deliverables

At the end of the IPA the following things have to be delivered to \\ch021012\SIM_MOD\Loris Isenegger:

- Source code of all the scripts
- Test documentation and test + mapping files used
- Documentation as described in 4.8
- All IPA related documentation (Journal, …)