**Technical Document**

**Pre-failure alert generation for equipment & cobble reduction based on data analytics and video analytics at BRM**

Document version: 0.1.0



**Department of Electrical Engineering and Computer Science,**

**IIT Bhilai, GEC Campus, Sejbahar, Raipur - 492015, (C.G.)**

**Technical Document for**

**Pre-failure alert generation for equipment & cobble reduction based on data analytics and video analytics at BRM**

Document version: 0.1.0

October – 2022

**Reviewed By**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Name of person**

**Designation, Department, Organisation**

**Approved By**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Name of person**

**Designation, Department, Organisation**

**Department of Electrical Engineering and Computer Science,**

**IIT Bhilai, GEC Campus, Sejbahar, Raipur - 492015, (C.G.)**

**Revision Records**

|  |  |  |
| --- | --- | --- |
| **Revision Records** | | |
| **Version number** | **Description** | **Document release date** |
| 0.1.0 | Initial release | October - 2022 |
|  |  |  |
|  |  |  |

**Preface**

The Technical document provides the basic information regarding the APIs, SDKs, system design and other logic to the developers and engineers in the project team.

**Terms of Use and Disclaimers**

The Technical document for ‘Pre-failure alert generation for equipment & cobble reduction based on data analytics and video analytics at BRM’ is released to provide the essential information regarding the APIs, SDKs, system design and other logic to the developers and engineers in the project team, according to the terms and conditions specified hereafter:

* The Technical Document shall not be reproduced or transmitted, partly or wholly, in electronic or print medium without the consent of the publishing authority.
* The information furnished in the Technical Document could be subject to modification and update.

**Copyright**

This Technical Document is protected by copyright. Any alteration or translation in any language of the Technical Documentation as a whole or parts of it is prohibited unless the Publishing Authority provides specific written prior permission.

The copyright notice - Copyright © Indian Institute of Technology Bhilai should not be removed.

**Table of Contents**

[1. Introduction 1](#_Toc116656064)

[**1.1 Scope of the document** 1](#_Toc116656065)

[**1.2 Document Overview** 1](#_Toc116656066)

[2. System Design and Overview 2](#_Toc116656067)

[**2.1 Server system design and their standards** 2](#_Toc116656068)

[**2.2 Server system connections and interaction between other systems** 2](#_Toc116656069)

[3. Storage Structure 3](#_Toc116656070)

[**3.1 File system design** 3](#_Toc116656071)

[**3.2 Raid and raid controller standards and design logic** 4](#_Toc116656072)

[**3.3 Data storage design** 5](#_Toc116656073)

[**3.3.1 Archive storage design** 7](#_Toc116656074)

[**3.4 Data management logics** 8](#_Toc116656075)

[4. Data Structures 9](#_Toc116656076)

[**4.1 PLC data format** 9](#_Toc116656077)

[**4.2 OPC standards** 9](#_Toc116656078)

[**4.3 Parquet data format** 9](#_Toc116656079)

[**4.4 Influx DB Structure** 9](#_Toc116656080)

[5. PLC Analytical System 16](#_Toc116656081)

[**5.1 Obtaining data from PLC signals** 16](#_Toc116656082)

[**5.2 Processing of Data** 16](#_Toc116656083)

[**5.3 Data Filtering and analysis** 22](#_Toc116656084)

[**5.4 Data conversion and archiving** 22](#_Toc116656085)

[**5.5 Data to time series database** 22](#_Toc116656086)

[6. Video Analytical System 22](#_Toc116656087)

[**6.1 GigE vision standards and design** 22](#_Toc116656088)

[**6.2 GigE feeds storage and analysis logic** 22](#_Toc116656089)

[Acronyms 23](#_Toc116656090)

**List of Figures**

[Figure 1 Sample Partition Structure 4](#_Toc116709161)

[Figure 2 Zpool status 4](#_Toc116709162)

[Figure 3/data partition hierarchy 5](#_Toc116709163)

[Figure 4 Sample /data hierarchy 6](#_Toc116709164)

[Figure 5 /archives partition hierarchy 7](#_Toc116709165)

[Figure 6 Sample /archive hierarchy 7](#_Toc116709166)

[Figure 7 JSON schema sample for the camera 13](#_Toc116709167)

[Figure 8 JSON schema sample for the camera 14](#_Toc116709168)

[Figure 9 JSON schema sample for our pre-defined events 14](#_Toc116709169)

[Figure 10 Sample statistics config file 19](#_Toc116709170)

[Figure 11 Sample select config file 21](#_Toc116709171)

**List of Tables**

[Table 1 cam\_feeds 10](#_Toc116656298)

[Table 2 sample data for /data 11](#_Toc116656299)

[Table 3 snsrs\_data 11](#_Toc116656300)

[Table 4 sensor\_data 12](#_Toc116656301)

[Table 5 abnormal\_pattern 12](#_Toc116656302)

[Table 6 Sample data for abnormal\_pattern 13](#_Toc116656303)

# 1. Introduction

## **1.1 Scope of the document**

## **1.2 Document Overview**

# 2. System Design and Overview

## **2.1 Server system design and their standards**

## **2.2 Server system connections and interaction between other systems**

# 3. Storage Structure

## **3.1 File system design**

We have decided to use ZFS as our file system, also I have designed the storage hierarchy of our feeds. I’ll be mentioning how file system, and other storage management logics will be working.

Partitioning of Drives: For production server we will setup the servers and drives on own. Though there will be a script/program which will help us to automate the server setup for the first time use.

Programs like fdisk, cfdisk etc. but I’ll be using sfdisk as it offer a non-interactive methods to partition our drive. Like we can first manually partition and structure our drive at first and then we can make a backup copy of that structure and later can be applied on the same.

I have tested that on our current server at IITB Electrical Lab on the HDD i.e. TOSHIBA MG04ACA200E (FP5B). Below are the current structure which is in raidz1.

|  |
| --- |
| label: gpt |
|  | label-id: E3B7FE9A-1618-4E8B-9EC7-4CA0340043DF |
|  | device: /dev/sdb |
|  | unit: sectors |
|  | first-lba: 34 |
|  | last-lba: 3907029134 |
|  |  |
|  | /dev/sdb1 : start= 2048, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=010F6E0C-BE91-42D9-9F1B-6885ACB3387F |
|  | /dev/sdb2 : start= 195315712, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=C183F619-DBD8-4728-82B8-BD369B5458D4 |
|  | /dev/sdb3 : start= 390629376, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=DFFCD8B4-7800-4784-9236-CFC86C7F0C0F |
|  | /dev/sdb4 : start= 585943040, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=B598E863-37D6-4D27-9299-D9D5B636A410 |
|  | /dev/sdb5 : start= 781256704, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=8E9C1E9E-FFE9-49F3-85CE-1AFEFF13EEC3 |
|  | /dev/sdb6 : start= 976570368, size= 195313664, type=0FC63DAF-8483-4772-8E79-3D69D8477DE4, uuid=851EF8D8-CE80-49C7-9750-40C3B9996FFD |

Using the command:

sfdisk -d /dev/sdb > IITB\_Sample\_Server\_partition\_with\_raidz1\_layout.sfd\_layout

And can be applied using:

sfdisk -d /dev/sdb < IITB\_Sample\_Server\_partition\_with\_raidz1\_layout.sfd\_layout

As sfdisk is a part of util-linux just like fdisk, so availability should be the same.

Mount points:

Sample partition structure below, parted using sfdisk.

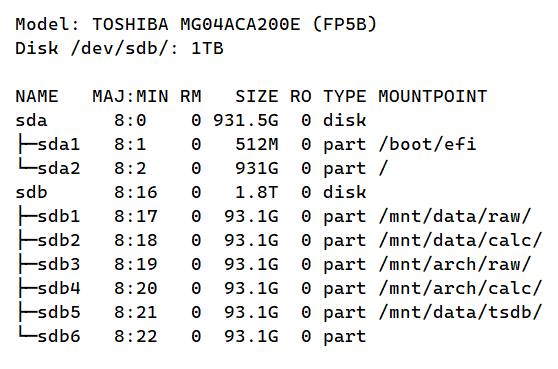


Figure Sample Partition Structure

Zpool status, formatted using zpool tool.

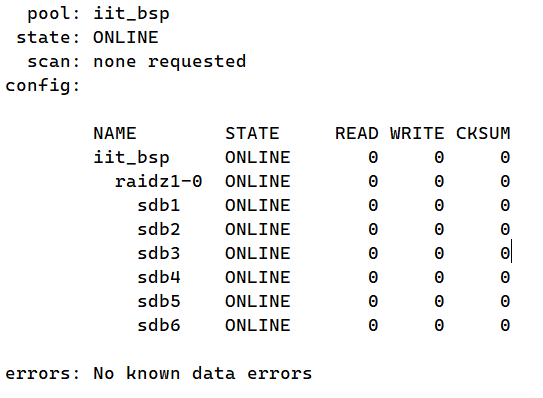


Figure Zpool status

## **3.2 Raid and raid controller standards and design logic**

## **3.3 Data storage design**

All the data for video streams will be mounted under /data. We will keep raw feeds for about 1 week (This may change in future depending on us) along with the processed (Calibrated) data. Then after we will keep it under /archives (maybe even in compressed format using ZSTD).

Note: This hierarchy design is only for video feeds not for data which is stored in our time series database. I may well make another partition and mount there not with /data, again to make data retrieval faster and keep TMDB safe in case if /data may go corrupt.

Figure /data partition hierarchy

I have divided the streams into zones and will categorise the respective cameras along with their zones. And as this path will be a live stream so there will be a script or a binary which will access the live feed from that camera with GigE/RTSP protocol in a GUI window with the IP:PORT rather than having a video file over there.

We will cut the video feeds from the live streams continuously like in an interval of 5 min and will be stored under /data/raw/unixdatetime/zone\_n\_cam\_n\_unixtimestart\_unixtimeend.mp4 for both the raw data (feeds) as well as for calibrated data but under /data/calibrated/\*.

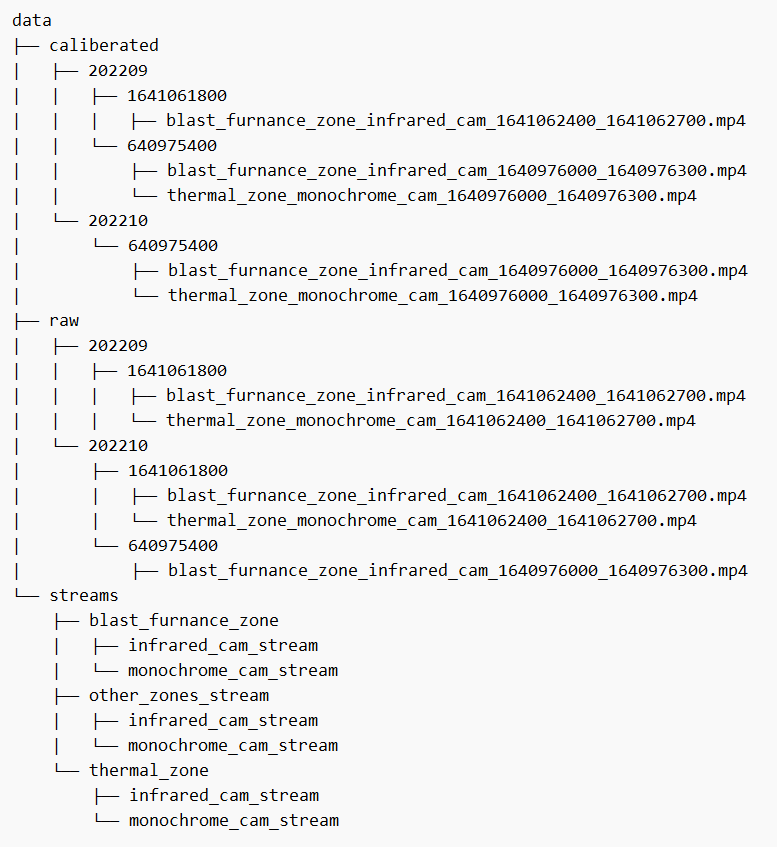


Figure Sample /data hierarchy

Some points

* There are many different ways in which we can define our storage hierarchy, for now, I have done it in the date-time manner in which all videos of the day1 are stored under the day1 folder and so on for dayn and then just the video files with 5min of interval and to query the videos we can go through the DateTime folder and the name of the file. The name of the file will contain the zone and camera from which it was taken.
* Other ways can be categorizing it in zones or the camera from which was taken and then the DateTime folder or just appending that in the file. (/data/zone1...n/datetime/video\_files or /data/cam1...n/datetime/video\_files).

### **3.3.1 Archive storage design**

As we will archive the raw as well as processed data there will be a separate partition for archives which is /archives. The raw data will probably be kept for 1 year at max and 2 years max for processed data (which may change in future).

Figure /archives partition hierarchy

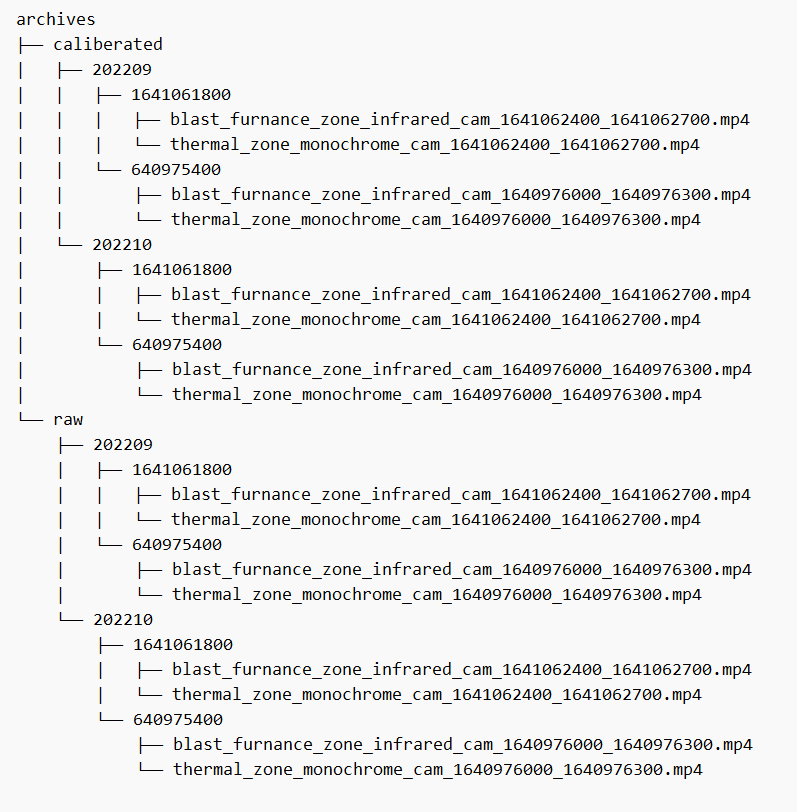


Figure Sample /archive hierarchy

## **3.4 Data management logics**

Storage Purging Logic

* I’m thinking to use the UNIX cron daemon for this regular purging process, as it was developed by AT&T and is part of the standard UNIX daemon so we can rely on that.
* Here is the basic cron expression:

# ┌───────────── minute (0 - 59)

# │ ┌───────────── hour (0 - 23)

# │ │ ┌───────────── day of the month (1 - 31)

# │ │ │ ┌───────────── month (1 - 12)

# │ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday;

# │ │ │ │ │ 7 is also Sunday on some systems)

# │ │ │ │ │

# │ │ │ │ │

# \* \* \* \* \* <command to execute>

* Let us take an example, we have millions of video feeds under /archive and we want to make sure the video feeds which are older than 2 years will be deleted regularly. So, we will make a script which will check whether there is any video or not if NULL or empty then return else check for videos whose DateTime is 2 years older than the current system time all this process will be run every day, so to this in cron:

0 0 \* \* \* <path\_to\_our\_deleting\_script\_or\_binary> // Every day at 12:00 AM

Or

@daily <path\_to\_our\_deleting\_script\_or\_binary> // Equivalent to above expression

* With cron daemon, we can have many combinations for our needs.
* crond also offers us to allow and deny the cron service for the users under /etc/cron.allow and /etc/cron.deny.
* And the same goes for /data, after 1 week or whatever time that we will define the raw and processed data will be moved safely to the archive in compressed format.

# 4. Data Structures

## **4.1 PLC data format**

## **4.2 OPC standards**

## **4.3 Parquet data format**

## **4.4 Influx DB Structure**

The InfluxDB will be used as our primary DBMS.

Some basic terminologies:

* Schema - How the data are organized in InfluxDB. The fundamentals of the InfluxDB schema are databases, retention policies, series, measurements, tag keys, tag values, and field keys. See Schema Design for more information.
* Retention Policy (RP) - Describes how long InfluxDB keeps data (duration), how many copies of the data to store in the cluster (replication factor), and the time range covered by shard groups (shard group duration). RPs are unique per database and along with the measurement and tag set define a series. When you create a database, InfluxDB creates a retention policy called autogen with an infinite duration, a replication factor set to one, and a shard group duration set to seven days.
* Timestamp - The date and time associated with a point. All time in InfluxDB is UTC but can be changed.
* Measurement - The part of the InfluxDB data structure that describes the data stored in the associated fields. Measurements are strings.
* Tag - The key-value pair in the InfluxDB data structure that records metadata. Tags are an optional part of the data structure, but they are useful for storing commonly-queried metadata; tags are indexed so queries on tags are performant. Query tip: Compare tags to fields; fields are not indexed.
* Tag key - The key part of the key-value pair that makes up a tag. Tag keys are strings and they store metadata. Tag keys are indexed so queries on tag keys are performant.
* Tag set - The collection of tag keys and tag values on a point.
* Tag value - The value part of the key-value pair that makes up a tag. Tag values are strings and they store metadata. Tag values are indexed so queries on tag values are performant.
* Field - The key-value pair in an InfluxDB data structure that records metadata and the actual data value. Fields are required in InfluxDB data structures and they are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant relative to tags.
* Field key - The key part of the key-value pair that makes up a field. Field keys are strings and they store metadata.
* Field set - The collection of field keys and field values on a point.
* Field value -The value part of the key-value pair that makes up a field. Field values are the actual data; they can be strings, floats, integers, or Booleans. A field value is always associated with a timestamp. Field values are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant.

InfluxDB Schema for /data/\* and /archive/\* feed records

For /data we will be having a bucket names cam\_feeds with a retention policy of about 1 week. Under cam\_feeds measurements of raw\_feeds and calc\_feeds will be there. We will be using feed\_id as a tag key so that it can be indexable like a primary key. The field key will be used as key=value pair to store our records like feed\_title, start\_time, end\_time, vid\_path, and from\_cam. After 1 week i.e., our retention policy, all data will be transferred from raw\_feeds > arch\_raw\_feeds and calc\_feeds > arch\_calc\_feeds.

|  |  |
| --- | --- |
| **Buckets** | cam\_feeds  (Retention policy = 1 week) |
| **Measurements** | raw\_feeds, calc\_feeds, arch\_raw\_feeds, arch\_calc\_feeds |
| **Tags Keys** | feed\_id (String, index) |
| **Field Keys** | * feed\_title (String) * start\_time (unixtime) * end\_time (unixtime) * vid\_path (String) * from\_cam (String) |

Table cam\_feeds

Let us take a sample data for only 1 raw feed for /data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **\_time** | **\_measurement** | **feed\_id** | **\_field** | **\_value** |
| 2022-01-01T00:00:00Z | raw\_feeds | 3eqenu3e3 | feed\_title | brm\_stand1\_therm\_cam\_1641062400\_1641062700.mp4 |
| 2022-01-01T00:00:00Z | raw\_feeds | 3eqenu3e3 | start\_time | 1641062400 |
| 2022-01-01T00:00:00Z | raw\_feeds | 3eqenu3e3 | end\_time | 1641062700 |
| 2022-01-01T00:00:00Z | raw\_feeds | 3eqenu3e3 | vid\_path | /data/raw/202209/640975400/ |
| 2022-01-01T00:00:00Z | raw\_feeds | 3eqenu3e3 | from\_cam | stand\_1\_thermal\_cam |

Table sample data for /data

Schema:

raw\_feeds, feed\_id=3eqenu3e3, feed\_title=brm\_stand1\_therm\_cam\_1641062400\_1641062700.mp4, start\_time=1641062400, end\_time=1641062700, vid\_path=/data/raw/202209/640975400/, from\_cam=stand\_1\_thermal\_cam

Same for calc\_feeds:

calc\_feeds, feed\_id=3eqenu3e3, feed\_title=brm\_stand1\_therm\_cam\_1641062400\_1641062700.mp4, start\_time=1641062400, end\_time=1641062700, vid\_path=/data/calc/202209/640975400/, from\_cam=stand\_1\_thermal\_cam

InfluxDB Schema for sensor data

Here we will be having all the data which are available from the installed sensors. Along with the other fields like \_time, sensor\_name, sensor\_zone, sensor\_type, and sensor\_value.

|  |  |
| --- | --- |
| **Buckets** | snsrs\_data  (Retention policy = 1 week) |
| **Measurements** | data |
| **Tags Keys** | snsr\_id (String, index) |
| **Field Keys** | * snsr\_name (String) * snsr\_zone (String) * snsr\_type (String) * snsr\_value (String) |

Table snsrs\_data

Let us take 1 sample data for sensor\_data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **\_time** | **\_measurement** | **snsr\_id** | **\_field** | **\_value** |
| 2022-01-01T00:00:00Z | data | 89d5a82s | snsr\_name | infrared\_sensor\_xyz\_inc\_bsp |
| 2022-01-01T00:00:00Z | data | 89d5a82s | snsr\_zone | brm\_stand\_16 |
| 2022-01-01T00:00:00Z | data | 89d5a82s | snsr\_type | infrared\_snsr |
| 2022-01-01T00:00:00Z | data | 89d5a82s | snsr\_value | 846.694546221949562 |

Table sensor\_data

Schema:

data, snsr\_id=89d5a82s, snsr\_name=infrared\_sensor\_xyz\_inc\_bsp, snsr\_zone=brm\_stand\_16, snsr\_type=infrared\_snsr, snsr\_value=846.694546221949562

As influxDB will automatically input a \_time whenever the record is entered, we will not have a different field for a timestamp and I’m assuming that there is no delay between the actual timestamp when a sensor has taken a value and the value of timestamp when the recorded is inserted, although ill make some tests.

InfluxDB Schema design for abnormal pattern records

We are going to store records which are abnormal or fall under our defined abnormal pattern category. This data will help us in future regarding the analysis and study of the data wherever, when and how the cobble or any other incident has happened.

|  |  |
| --- | --- |
| **Buckets** | abnormal\_pattern  (Retention policy = 1 week) |
| **Measurements** | data |
| **Tags Keys** | patt\_id (String, index) |
| **Field Keys** | * patt\_type (String) * patt\_snsr (String) * patt\_zone (String) * patt\_occ\_at (unixtime) * patt\_occ\_till (unixtime) |

Table abnormal\_pattern

Let us take 1 sample data for abnormal\_pattern:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **\_time** | **\_measurement** | **pattern\_id** | **\_field** | **\_value** |
| 2022-01-01T00:00:00Z | data | 5sd5sd68 | patt\_type | cobble\_med\_brm |
| 2022-01-01T00:00:00Z | data | 5sd5sd68 | patt\_snsr | infrared\_snsr |
| 2022-01-01T00:00:00Z | data | 5sd5sd68 | patt\_zone | brm\_stand\_16 |
| 2022-01-01T00:00:00Z | data | 5sd5sd68 | patt\_occ\_at | 1641062400 |
| 2022-01-01T00:00:00Z | data | 5sd5sd68 | patt\_occ\_till | 1641062700 |

Table Sample data for abnormal\_pattern

Schema:

data, patt\_id=89d5a82s, patt\_type=cobble\_med\_brm, patt\_zone=brm\_stand\_16, patt\_occ\_at=1641062400, patt\_occ\_till=1641062400

Workaround to relate records with InfluxDB

As I have mentioned that influxDB is not an RDBMS and because of that relations aren’t possible with influxDB. I was thinking to make a separate index for our available camera, sensors, and defined patterns (events) in our schema based on JSON. This will help us to have an index of those along with their IDs which are also tag keys in our influxDB records.

JSON schema sample for the camera:

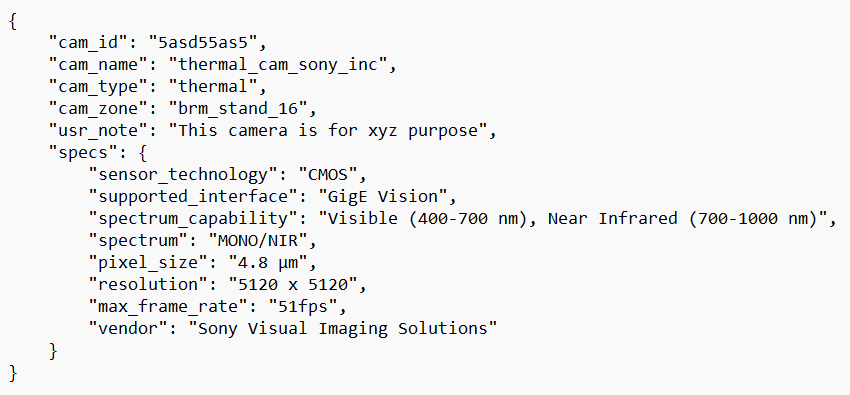


Figure JSON schema sample for the camera

JSON schema sample for our available sensors:

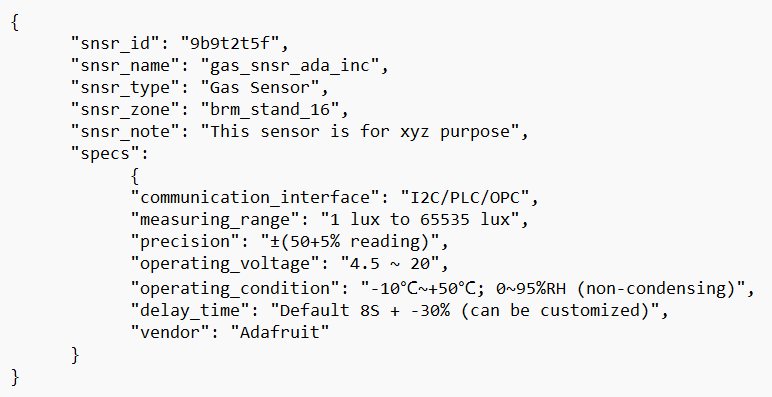


Figure JSON schema sample for the camera

JSON schema sample for our pre-defined events:



Figure JSON schema sample for our pre-defined events

Note: If we need to get more data about the sensor, camera, and more about events we can get it from these indexes using the IDs which are used as tags in our influxDB.

Security: In influxDB, we have token/password-based authentication for security, but to keep these index data secure we can limit permissions for the user using the UNIX system permission concept also we can encrypt these data using Encryption methods like AES. Encryption, Integrity, Security etc. are very important to us and I’ll write and explain more about them later in the other document.

# 5. PLC Analytical System

## **5.1 Obtaining data from PLC signals**

## **5.2 Processing of Data**

For conversion of CSV/TSV to Apache Parquet

As we have decided to use apache’s parquet instead of CSV/TSV which is slow and uses more storage compared to apache parquet which is an open-source, column-oriented data file format designed for efficient data storage and retrieval. This tool will allow us to convert our existing or old CSV/TSV data files to apache’s parquet with various other configurable parameters in a config file.

Working: It uses the native implementation of apache’s arrow and parquet in rust to read, decode and convert it. Using a config file which will be read and decoded by a shell script. It reads the users defined configuration from the *config.config* file and parse it as parameters to the converter.

Config.config structure: There can be multiple config files for multiple sensors or as per requirements, but the program will try to read from .configs/config.config.

Sample config file:

# Config file for the CSV/TSV to parquet conversion tool.

# The name of the config file.

NAME="bsp\_zone1\_snsr\_1-9"

# The description of the config file.

DESCRIPTION="This config is used for xyz purpose at BSP."

# Path to the csv file

PATH\_TO\_CSV="abc.csv"

# Path where the parquet file should be saved

PATH\_TO\_PQT="abc.parquet"

# Set the CSV file's column delimiter as a byte character [default: ,]

#DELIMITER="'\t'"

# created\_by tag for the file

CREATED\_BY="BSP\_BRM"

# The number of records to infer the schema from.

#MAX\_READ\_RECORDS="100"

# If the CSV file contains header

#HEADER=true

# Sets flag to enable/disable dictionary encoding for any column

DICTIONARY=true

# Compression

# Set the compression [possible values: uncompressed, snappy, gzip, lzo, brotli, lz4, zstd]

#COMPRESSION="uncompressed"

# Sets encoding for any column.

# [possible values: plain, rle, bit-packed, delta-binary-packed, delta-length-byte-array, delta-byte-array, rle-dictionary]

#ENCODING="plain"

# Sets data page size limit

#DATA\_PAGE\_SIZE\_LIMIT=""

# Sets dictionary page size limit

#DICTIONARY\_PAGE\_SIZE\_LIMIT=""

# Sets write batch size

#WRITE\_BATCH\_SIZE=""

# Sets max size for a row group

#MAX\_ROW\_GROUP\_SIZE=""

# Sets flag to enable/disable statistics for any column [possible values: none, chunk, page]

#STATISTICS="none"

# Sets max statistics size for any column. Applicable only if statistics are enabled

#MAX\_STATISTICS\_SIZE=""

# Print the schema as output in the terminal

SCHEMA=true

Fields

* Name: The name of the config file. As there can be multiple config files, naming it will help us to differentiate.
  + String: A string value.
* Description: Along with the name there can be a small description for the file where the purpose and other things can be described.
  + String: A string value.
* Path\_to\_csv: The path where the CSV/TSV file is located, should be inside ‘ ‘.
  + String: A string value.
* Path\_to\_pqt: The path where you want to save the parquet file, should be inside ‘ ‘. Also, the name of the file along with the extension (.parquet) should be in the path.
  + String: A string value
* created\_by: Add created\_by tag for the parquet file.
  + String: A string value.
* max\_read\_records: The maximum number of records to be read from the CSV/TSV file.
  + usize: should be an Unsigned Integer of value, usize types depend on the architecture of the computer your program is running on, which is denoted in the table as “arch”: 64 bits if you’re on a 64-bit architecture and 32 bits if you’re on a 32-bit architecture.
* header: Whether the CSV file contains the header or not.
  + Bool: Either True or False.
* dictionary: Sets flag to enable/disable dictionary encoding for any column.
  + Bool: Either True or False.

Note: The raw signals data which was exported directly from the ibaAnalyser as CSV needs to have same-length records else it will throw an uneven record error and the file will not be converted as parquet.

Let’s say there is a sample CSV file:

col1, col2, col3, col4, col5

val1, val2, val3, val4

val5, val6, val7, val8

Here we can see that our CSV have 5 columns but the values for col5 are NULL and can cause an error while converting. We have noticed that this was happening in all of the CSV files that we exported from the ibaAnalyser. To overcome from this we need to make sure the CSV files have same-length records and for that, we have a tool (xsv) which is installed on our server which can force the file to have the same-length record either by padding or truncating them.

Documentation for csvtool: for indexing, slicing, analysing, splitting and joining CSV

This program will be used by us to analyse the CSV/TSV files like mean, avg, median, sum, max/min, cardinality and other stats. Also will allow us to modify the CSV/TSV files like re-ordering the columns, and rows, slicing, joining, splitting etc.

As we have noticed that we have a large number of records in our CSV files, so we need a fast and efficient tool to read, organise and perform other operations in our CSV file for that, we will be using a tool name xsv.

xsv will allow us to create an index before performing any operations on the file so that we can perform slicing, splitting, and gathering statistics much faster. It will create an index file on the same path as input.csv.idx The index will be automatically used by commands that can benefit from it. If the original CSV data changes after the index is made, commands that try to use it will result in an error (you have to regenerate the index before it can be used again).

To create an index for the CSV file:

*xsv index path\_to\_csv*

xsv can also generate some statistics like sum, min, max, mean, stddev, median, mode, cardinality etc

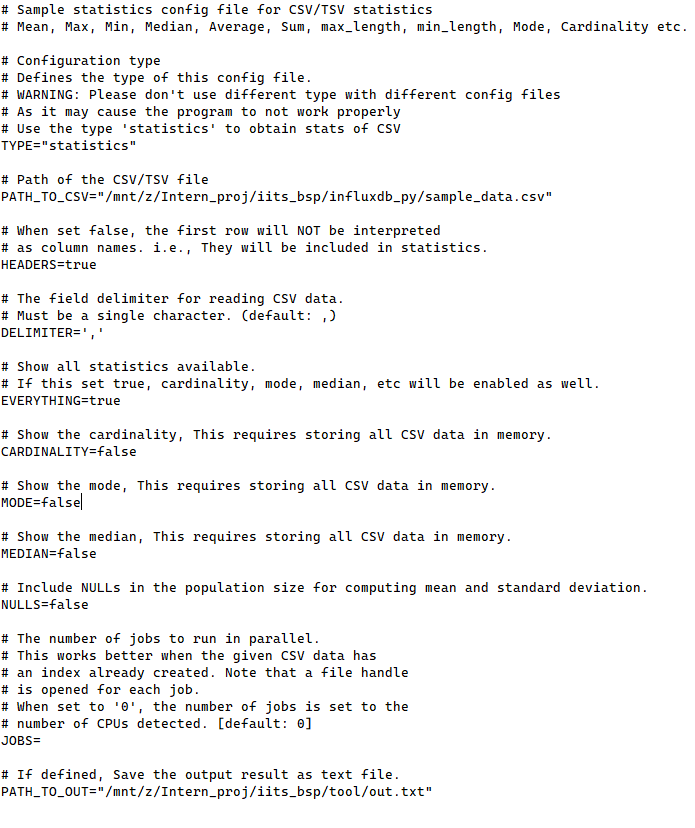


Figure Sample statistics config file

Fields

* TYPE: A String which denotes the work which needs to be done by the program, please don’t use different types with different defined work.
* PATH\_TO\_CSV: A path to the CSV/TSV file which will be used to generate stats.
* HEADERS: Takes a Boolean value, if set to false then the first row will be interpreted as statistics, not as columns.
* DELIMITER: A char value which is the field delimiter for the CSV/TSV data.
* EVERYTHING: Show all statistics available. If this is set true, cardinality, mode, median, etc. will be enabled as well.
* CARDINALITY: Show the cardinality, this requires storing all CSV data in memory.
* MODE: Show the mode, this requires storing all CSV data in memory.
* MEDIAN: Show the median, this requires storing all CSV data in memory.
* NULLS: Include NULLs in the population size for computing mean and standard deviation.
* JOBS: The number of jobs to run in parallel. This works better when the given CSV data has an index already created. Note that a file handle is opened for each job. When set to '0', the number of jobs is set to the number of CPUs detected. [Default: 0]
* PATH\_TO\_OUT: If defined, Save the output result in a text file.

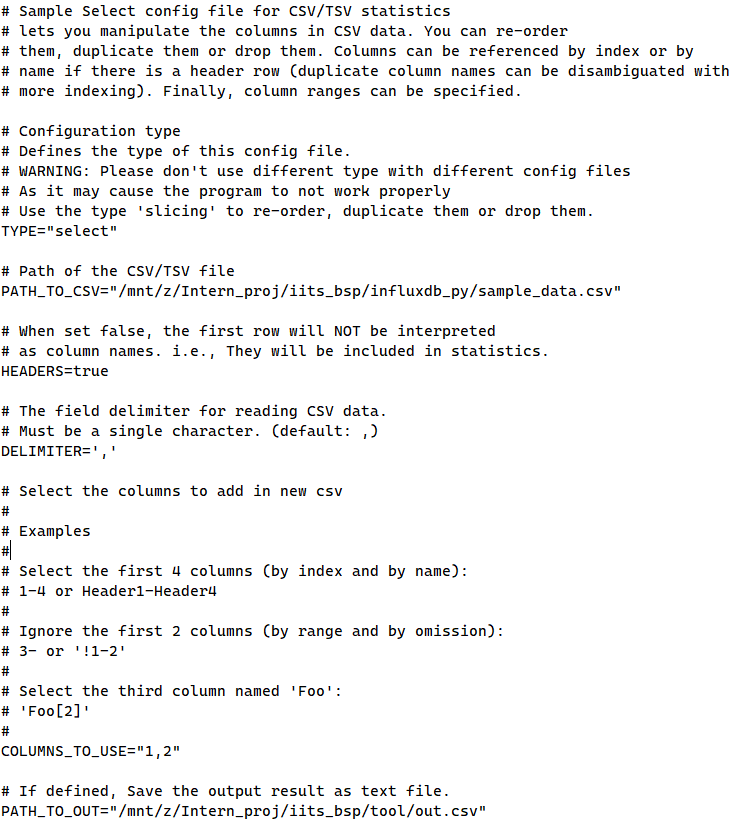


Figure Sample select config file

Fields

* TYPE: A String which denotes the work which needs to be done by the program, please don’t use different types with different defined work.
* PATH\_TO\_CSV: A path to the CSV/TSV file which will be used to generate stats.
* HEADERS: Takes a Boolean value, if set false then the first row will be interpreted as statistics, not as columns.
* DELIMITER: A char value which is the field delimiter for the CSV/TSV data.
* COLUMNS\_TO\_USE: Selects the columns of the file which should be used.
  + Select the first 4 columns (by index and by name): 1-4 or Header1-Header4
  + Ignore the first 2 columns (by the range and by omission): 3- or '!1-2'
  + Select the third column named 'Foo': 'Foo[2]'
* PATH\_TO\_OUT: If defined, Save the output result as a new CSV file.

### **5.3 Data Filtering and analysis**

### **5.4 Data conversion and archiving**

### **5.5 Data to time series database**

# 6. Video Analytical System

## **6.1 GigE vision standards and design**

## **6.2 GigE feeds storage and analysis logic**

# Acronyms

**PLC Programmable Logic Controller**

**OPC Open Protocol Communication**

**CSV Comma Separated Values**

**UTF8 UCS (Unicode) Transformation Format**

**ZSTD Z Standard**

**RP Retention Policy**

**ZFS Zettabyte File System**

**RAID Redundant Array of Inexpensive Disks**

**TSDB Time Series Data Base**