EPSON

VC Evaluation

Vibration Measurement System

- User's Guide

Notice of Document

Evaluation board/kit and development tool important notice

1. This evaluation board/kit or development tool is designed for use for engineering evaluation, demonstration, or development purposes only. Do not use it for other purposes. It is not intended to meet the requirements of design for finished products.
2. This evaluation board/kit or development tool is intended for use by an electronic engineer and is not a consumer product. The user should use it properly and in a safe manner. Seiko Epson does not assume any responsibility or liability of any kind of damage and/or fire caused by the use of it. The user should cease to use it when any abnormal issue occurs even during proper and safe use.
3. The part used for this evaluation board/kit or development tool may be changed without any notice.

NOTICE : PLEASE READ CAREFULLY BELOW BEFORE THE USE OF THIS DOCUMENT

The content of this document is subject to change without notice.

1. This document may not be copied, reproduced, or used for any other purposes, in whole or in part, without the consent of Seiko Epson Corporation("Epson").
2. Before purchasing or using Epson products, please contact with our sales representative for the latest information and be always sure to check the latest information published on Epson's official web sites and sources.
3. Information provided in this document such as application circuits, programs, usage, etc., are for reference purpose only. Please use the application circuits, programs, usage, etc. in the design of your equipment or systems at your own responsibility. Epson makes no guarantees against any infringements or damages to any third parties' intellectual property rights or any other rights resulting from the information. This document does not grant you any licenses, intellectual property rights or any other rights with respect to Epson products owned by Epson or any third parties.
4. Epson is committed to constantly improving quality and reliability, but semiconductor products in general are subject to malfunction and failure. In using Epson products, you shall be responsible for safe design in your products; your hardware, software and systems are designed enough to prevent any harm or damages to life, health or property even if any malfunction or failure might be caused by Epson products. In designing of your products with using Epson products, please be sure to check and comply with the latest information regarding Epson products (this document, specifications, data sheets, manuals, Epson's web site, etc.). When using the information included in the above materials such as product data, chart, technical contents, programs, algorithms and application circuit examples, you shall evaluate your products both in stand-alone basis and within your overall systems. You shall be solely responsible for deciding whether or not to adopt and use Epson products.
5. Epson has prepared this document and programs provided in this document carefully to be accurate and dependable, but Epson does not guarantee that the information and the programs are always accurate and complete. Epson assumes no responsibility for any damages which you incurred by due to misinformation in this document and the programs.
6. No dismantling, analysis, reverse engineering, modification, alteration, adaptation, reproduction, etc., of Epson products is allowed.
7. Epson products have been designed, developed and manufactured to be used in general electronic applications (office equipment, communications equipment, measuring instruments, home electronics, etc.) and applications individually listed in this document ("General Purpose"). Epson products are NOT intended for any use beyond the General Purpose that requires particular/higher quality or reliability in order to refrain from causing any malfunction or failure leading to harm to life, health or serious property damage or severe impact on society, including, but not limited to listed below. Therefore, you are advised to use Epson products only for the General Purpose. Should you desire to buy and use Epson products for the particular purpose other than the General Purpose, Epson makes no warranty and disclaims with respect to Epson products, whether express or implied, including without limitation any implied warranty of merchantability or fitness for any particular purpose.
   [Particular purpose]
   Space equipment (artificial satellites, rockets, etc.)
   Transportation vehicles and their control equipment (automobiles, aircraft, trains, ships, etc.)
   Medical equipment (other than applications individually listed in this document) / Relay equipment to be placed on sea floor
   Power station control equipment / Disaster or crime prevention equipment / Traffic control equipment / Financial equipment
   Other applications requiring similar levels of reliability as the above
8. Epson products listed in this document and our associated technologies shall not be used in any equipment or systems that laws and regulations in Japan or any other countries prohibit to manufacture, use or sell. Furthermore, Epson products and our associated technologies shall not be used for developing military weapons of mass destruction, military purpose use, or any other military applications. If exporting Epson products or our associated technologies, you shall comply with the Foreign Exchange and Foreign Trade Control Act in Japan, Export Administration Regulations in the U.S.A (EAR) and other export-related laws and regulations in Japan and any other countries and follow the required procedures as provided by the relevant laws and regulations.
9. Epson assumes no responsibility for any damages (whether direct or indirect) caused by or in relation with your non-compliance with the terms and conditions in this document.
10. Epson assumes no responsibility for any damages (whether direct or indirect) incurred by any third party that you assign, transfer, loan, etc., Epson products.
11. For more details or other concerns about this document, please contact our sales representative.
12. Company names and product names listed in this document are trademarks or registered trademarks of their respective companies.

2022.08

Trademark

- Raspberry Pi’s a trademark of Raspberry Pi Foundation.
- Microsoft and Windows are trademarks of the Microsoft group of companies.
- Epson is a registered trademark of Seiko Epson Corporation.
- Other product names are trademarks or registered trademarks of the respective companies.

Table of Contents

Revision History

Rev. No.	Rev. Date	Page	Rev. Contents
20250331	2025/3/31	ALL	First edition Corresponding to the release of MSG006-001a_v1.0.0

1. Related Documents

1. “Setup Manual for Vibration Measurement System using Raspberry Pi Products” Rev.20250331
2. “Operation Manual for Vibration Measurement System using Raspberry Pi Products” Rev.20250331
3. “VC Evaluation Library - Reference Manual” Rev.20250221
4. “Monitoring Application for Vibration Measurement System – User’s Guide” Rev.20250331
5. “ADRSRU4 Relay control expansion board for Raspberry Pi,” Bit Trade One Co., Ltd. (https://bit-trade-one.co.jp/en/product/module/adrsru/)

2. Introduction

This user guide explains the following aspects of the “VC-Based Vibration Measurement System”:

- Specifications
- Setup Procedure
- Execution Method
- Developer Guide

The “VC-Based Vibration Measurement System” (hereafter referred to as “this application”) is an application that combines the “Vibration Measurement System using Raspberry Pi” (hereafter “Raspy Logger”) and the “VC Evaluation Library.”

This application processes measurement data from acceleration sensors via the Raspi Logger and calculates VC levels using the VC Evaluation Library, then outputs the results.

By installing the “BitTradeOne Relay Control Expansion Board ADRSRU4” (hereafter “Relay Control Board”) on the Raspberry Pi, it is possible to control relays based on vibration levels.

In addition to sending status messages via the MQTT broker of the Raspi Logger, this application can also send VC evaluation results as MQTT messages. These messages can be viewed in a web browser on a PC using the “Vibration Measurement System Monitoring Application” (hereafter “Monitoring App”).

Since the core execution mechanism is based on the Raspi Logger, the execution and measurement methods follow the same procedures.

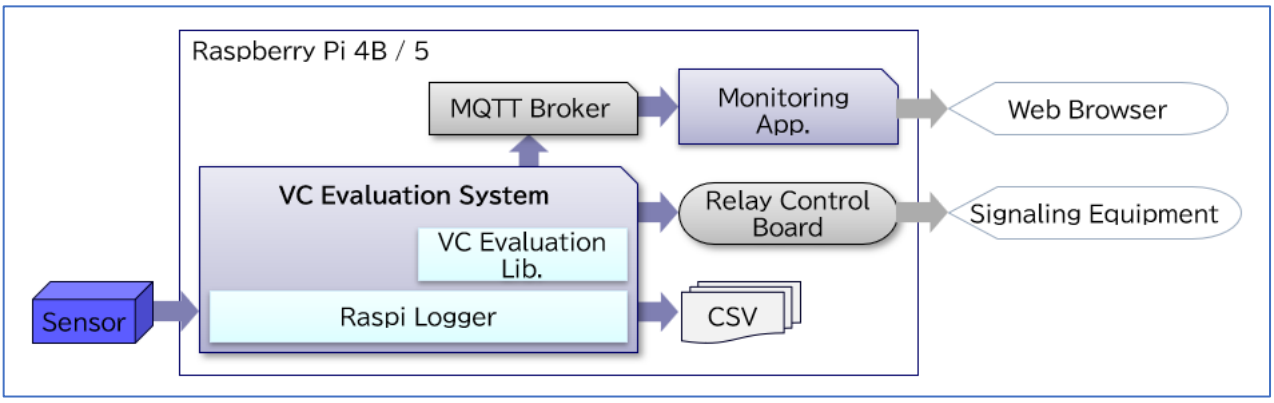

Figure 2-1 Overall System Configuration

3. Specifications

3.1. Verified Operating Environment

- Raspberry Pi 4B
- Raspberry Pi 5

3.2. Supported Sensors

- M-A352AD
- M-A552AR

Note: This application does not support M-A342VD and M-A542VR.

3.3. Application Configuration Parameters

Since this application is based on Raspi Logger, its configuration parameters are inherited. For details, refer to Related Document 2. The following are parameters specific to this application:

Table 3-1

Parameter Name	Description	Allowed Values	Notes
VC_FFT_SIZE	Number of FFT points for VC evaluation	Even numbers between 2000~20000	*1
VC_EXE_SIZE	Number of data points used in VC evaluation	Integer divisible by VC_FFT_SIZE	*1
VC_AVG_SIZE	Averaging count for VC average output	Integer between 10-50	*1
VC_LOG_SINGLE	Whether to log SINGLE evaluation result	True, False	
VC_LOG_AVERAGE	Whether to log AVERAGE evaluation result	True, False	
VC_OUTPUT_RAW	Whether to output raw measurement data as CSV	True, False	
VC_CONTROL_GPIO	Whether to enable GPIO control	True, False	

- *1 : See Related Document 3 for details.

3.4. Execution Specifications

The execution method of this application follows the Raspi Logger. Refer to Related Document 2 for details.

3.5. Input Specifications

This application supports measurement using multiple M-A352AD/M-A552AR sensors connected via USB. Only a sampling rate of **1000 SPS** is supported.

Note: M-A342VD and M-A542VR sensors are not supported. If connected, an error will occur, and the application will terminate.

Note: If the sampling rate is set to anything other than 1000 SPS via the A352_SPS parameter, an error will occur, and the application will terminate.

3.6. Output Specifications

In addition to the measurement functions of Raspi Logger, this application performs VC evaluation calculations and outputs the following types of data:

1. Measurement Data File
2. Measurement Info File
3. VC Evaluation Data File
4. VC Evaluation Level (to log file)
5. VC Evaluation Level (to GPIO)
6. Status Messages
7. Log File

3.6.2. Measurement Data File

The measurement data file records the raw data collected during measurement. This file is generated according to the specifications of the Raspi Logger. If the configuration parameter `VC_OUTPUT_RAW` is set to **`False`**, this file will **not** be generated.

3.6.3. Measurement Information File

The measurement information file contains metadata related to the measurement session. In addition to the standard output items from the Raspi Logger, this application adds its own configuration parameters to the file.

3.6.4. VC Evaluation Data File

This application performs VC evaluation on the measurement data and outputs the results as CSV files. The VC Evaluation Library performs two types of calculations: **SINGLE** and **AVERAGE**, which are output to separate files.

VC Evaluation File Names:

- SINGLE : “[Sensor Model]_[Sensor Serial]_vccalc_sgl.csv”
- AVERAGE : “[Sensor Model]_[Sensor Serial]_vccalc_avg.csv”

VC Evaluation File Layout (common to SINGLE and AVERAGE)

- Data Fields (119 columns per row):
  - Serial Number
  - Timestamp
  - VC Evaluation Level (OA, A, B, C, D, E, F, G)
  - For each of the following axes: Composite, X Axis, Y Axis, Z Axis:
    - Peak Velocity (mm/s)
    - Peak Frequency (Hz)
    - 1/3 Octave Band Velocity Results (27 values)
- Header Rows (2 rows)
  - First: No, Time, VC-Level, Composite, X Axis, Y Axis, Z Axis
  - Second: Peak Velocity, Peak Frequency, 1.00 Hz, 1.26 Hz, ⋯, up to 27 bands (1/3 octave band)
- Data Rows (from 3rd row onward)

No	Time	VC-Level	Composite				
			Peak Velocity (mm/s)	Peak Frequency (Hz)	1.00 (Hz)	1.26 (Hz)	⋯
1	2025/03/01 10:00:00	C	0.011457	10.08	0.000321	0.000221	⋯

2	2025/03/01 10:00:01	C	0.011258	10.08	0.000307	0.000225	…
…	…	…	…	…	…	…	…

Figure 3-1 Example of VC Evaluation Data File

3.6.5. VC Evaluation Level (Output to Log File)

Among the results of the VC evaluation calculation, the VC evaluation level is output to the log file.

If the configuration parameters VC_LOG_SINGLE or VC_LOG_AVERAGE is set to **False**, the corresponding results will **not** be output.

3.6.6. VC Evaluation Level (Output to GPIO)

Based on the VC evaluation level, this application controls GPIO pins and outputs signals to the relay control board. If the configuration parameter VC_CONTROL_GPIO is set to **False**, this control will not be performed.

GPIO control follows the specifications of the relay control board.

Table 3-2 ADRSRU4 GPIO Specifications

Relay Number	GPIO Pin Number
1	4
2	17
3	27
4	22

Accordingly, the relay board is controlled for each VC evaluation level as shown below.

Table 3-3 Relay Control by VC Evaluation Level

Relay Number	VC Evaluation Level							
	OA	A	B	C	D	E	F	G
1	ON							
2		ON						
3			ON					
4				ON	ON	ON	ON	ON

3.6.7. Status Messages

This application sends MQTT messages containing the VC evaluation level and FFT results.

These messages can be viewed in a web browser on a PC using the Monitoring Application.
For details, refer to Related Document 4.

Table 3-4 Status Message Specifications

Message Type	Topic Name	Message Content
Sensor VC Level	logger/$LOG/sensor/$MDL/$SNO/vc	level: “OA”|”A”|”B”|”C”|”D”|”E”|”F”|”G”
Sensor FFT	logger/$LOG/sensor/$MDL/$SNO/fft	value: [0.000282,0.000124,0.000162,…]

- $LOG: Value of LOGGER_ID in the configuration file
- $MDL: Sensor model name
- $SNO: Sensor serial number

- The `value` field in the FFT message contains an array of 27 values calculated as "Composite".
- Frequency values are not included in the message.
- All messages are in JSON string format and include a `timestamp` field in the format: `"timestamp": "yyyy/mm/dd hh:mm:ss.nnnnnn"`

3.6.8. Log File

This application outputs log files in accordance with the specifications of the Raspi Logger.

The following information is output specifically by this application:

- File names of VC evaluation data files
- VC evaluation levels

3.6.9. Output Folder

All files output by this application are saved in folders named according to the [**measurement date and time**], following the same specifications as the Raspi Logger.

4. Setting Up on Raspberry Pi

This section describes the procedure for setting up this application on a Raspberry Pi.

4.1. File and Folder Structure

After extracting the downloaded ZIP file, the file and folder structure is as follows:

```
Extracted Folder
├──── bin                                  # Configuration files required to build the runtime
├──── pyproject.toml                       # Python project configuration file
├──── src                                  # Python source code of this application
├──── sub                                  # Installation packages for referenced libraries
│     ├──── raspi_logger-1.2.0-py3-none-any.whl     # Raspi Logger installation file
│     └──── vc_calculation-1.0.0-py3-none-any.whl   # VC Evaluation Library installation file
└──── .env.default                         # Template for the application's configuration file
```

Figure 4-1 File and Folder Structure

4.2. Preparation

Before starting the setup, refer to Related Document 1 and complete the following preparations:

- Connect the Raspberry Pi and PC to the same network
- Ensure the Raspberry Pi has internet access
- Install an MQTT broker (e.g.,mosquitto) on the Raspberry Pi

This guide assumes the following:

- Raspberry Pi username: `pi`
- Fixed IP address: `192.168.1.52`

If you use different values, please adjust the commands accordingly.

4.3. Transfer ZIP File to Raspberry Pi

Run the following command to transfer the downloaded ZIP file to the Raspberry Pi:

- `scp MSG006-001a.zip pi@192.168.1.52:.`

The file will be transferred to the home directory of the `pi` user.

4.4. Install the Application on Raspberry Pi

Follow these steps to install the application:

1. Log in to the Raspberry Pi:
   - `ssh pi@192.168.1.52`
2. Create the installation directory:
   - `sudo mkdir /app` (* If /app does not exist)
   - `sudo chown pi:pi /app` (* Change ownership)

- `mkdir -p /app/MSG006-001a`

3. Extract the ZIP file:
   - `cd /app/MSG006-001a`
   - `unzip ~/MSG006-001a.zip`
4. Create a Python virtual environment for running the application:
   (Use system-installed GPIO modules by specifying an additional option)
   - `python -m venv --system-site-packages venv`
   - `source venv/bin/activate`
5. Upgrade `pip` to the latest version:
   - `pip install -U pip`
6. Install the required libraries:
   - `pip install . sub/*.whl`
7. Install service registration files to the OS for executing each application including this application as needed:
   - For this application
     `sudo cp bin/vc_calc_app@.service /etc/systemd/system`
   - For Raspi Logger
     `sudo cp bin/logger@.service /etc/systemd/system`
   - For Raspi Logger hardware monitor
     `sudo cp bin/hwmonitor.service /etc/systemd/system`
8. Reload systemd to recognize the new services:
   - `sudo systemctl daemon-reload`

Note: These service files assume the Python virtual environment is located at /app/MSG006-001a.
If you use a different path, modify the following line in each file accordingly:

`ExecStart=/app/MSG006-001a/venv/bin/python -m` *(the rest of the lines varies depending on the application)*

4.5. Application Configuration

Copy the configuration file template and create the actual configuration file:

- `cp .env.default .env`

Edit the `.env` file as needed using a text editor.

Refer to Section 3.3 Application Configuration Parameters for details on each setting.

5. Running the Application

The execution method for this application follows the same procedure as the Raspi Logger.

You can run the application in the following ways:

- Register the application to start automatically at OS boot
  - `sudo systemctl enable vc_calc_app@auto.service`
- Start the application using the measurement time specified in the configuration file
  - `sudo systemctl start vc_calc_app@manual.service`
- Start the application by specifying the measurement time (in seconds)
  - `sudo systemctl start vc_calc_app@[measurement_time].service`
  - * Replace [`measurement_time`] with the desired number of seconds.

Alternatively, run the application directly using the `Python` command

- `python -m vc_calc_app [measurement_time]`

For more details, refer to Related Document 2.

6. Appendix : Developer Guide

This chapter serves as a developer guide, providing information necessary not only for adding or modifying features of this application, but also for developing derivative applications based on Raspi Logger.

- Building the development environment
- Customizing Raspi Logger
- Building the development environment

6.1. Building the development environment

Assumes the ZIP file has been extracted to a folder on your local PC.

6.1.1. Creating a Python Virtual Environment

Launch PowerShell/Terminal and create Python virtual environment in a folder extracted the ZIP file.

For Windows:

- Launch PowerShell and execute the following command.
  - `py -3.11 -m venv venv`
  - `venv\scripts\activate.ps1`

For macOS/Linux:

- Launch Terminal and execute the following command.
  - `python3.11 -m venv venv`
  - `source venv/bin/activate`

6.1.2. Installing Packages

Install the packages used by this application.

For Windows (PowerShell):

- `python -m pip install -U pip`
- `pip install -e .`

For macOS/Linux (Terminal):

- `pip install -U pip`
- `pip install -e .`

The source code of this application is installed into the virtual environment in editable mode.

To install packages used only for development and testing:

For Windows:

- `pip install -e .[dev,test]`

For MacOS/Linux:

- `pip install -e ".[dev,test]"`

6.2. Raspi Logger Customization Guide

6.2.1. Overview of Raspi Logger Program

Raspi Logger is a program that measures vibration data using USB-connected sensors on a Raspberry Pi and saves the data to CSV files. The process follows this general flow:

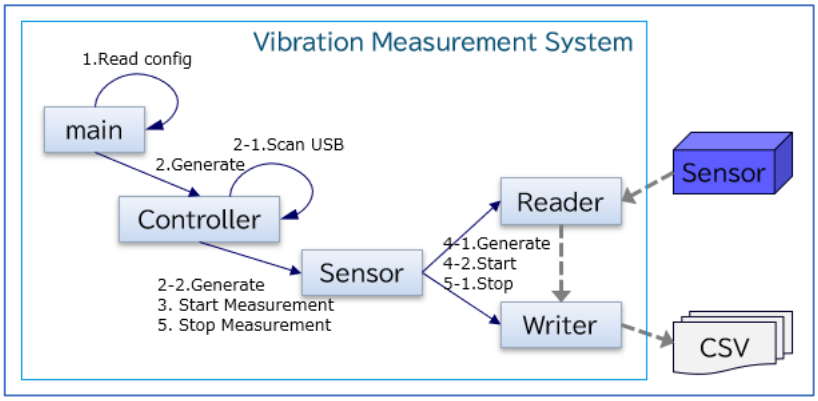

Figure 6-1 RasPi Logger Sequence

1. The program starts and reads configuration values from a file.
2. The program scans connected USB and generates Sensor object controlling a sensor according to the found model name.
3. The program orders Sensor object to start measurement.
4. The Sensor object generates Reader process to read data from sensor and Writer process to output the read data, thereby performing the measurement operation.
5. The program stops the measurement either at a preconfigured time or in response to an external signal.

If multiple sensors are connected, a set of Sensor, Reader, and Writer objects is created for each. For example, if four sensors are connected, the objects are structured as shown in the following figure.

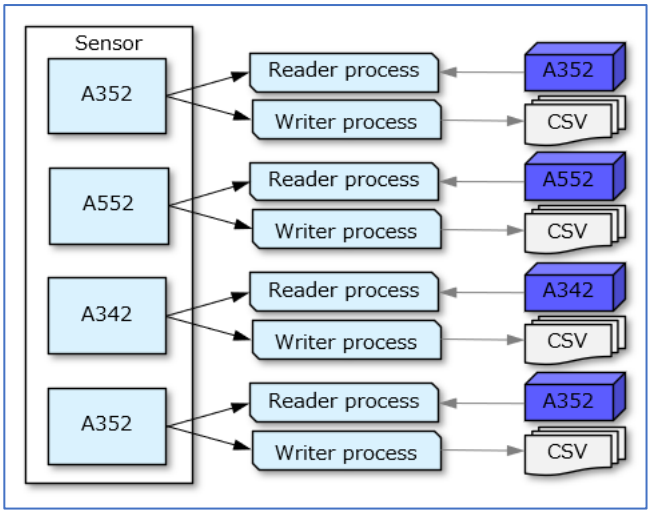

Figure 6-2 Raspi Logger Object Structure

6.2.2. Customization Overview

RasPi Logger is a Python program that operates using multiprocessing, and it is equipped with a mechanism that allows key objects to be replaced to support extensibility.

Key objects are generated from a single **Factory object**, which can be replaced to use extended versions of those objects.

This follows the **Abstract Factory** design pattern.

Since the Factory object can be passed to the main function, you can replace the Factory object by creating an extended version before calling main and passing it as an argument.

6.2.3. `logger.LoggerFactory` Class

This is the interface class for the Factory object that generates key components in Raspi Logger. The default implementation is `logger.core.LoggerFactoryImpl`.

To customize, inherit from `LoggerFactoryImpl` and override the necessary methods:

- `create_configurator()`: Generates a Configurator object.
- `create_A342()`: Creates an object to control A342 sensors.
- `create_A352()`: Creates an object to control A352 sensors.
- `create_reader_process()`: Creates the Reader process.
- `create_writer_process()`: Creates the Writer process.

6.2.3.1. `create_configurator()` Method

This method generates a `logger.core.Configurator` object, which checks the configuration file and creates a `logger.core.Config` object that holds the configuration values.

If you want to add new configuration values, define an extended `Configurator` class, and modify this method to return the object of that class.

6.2.3.2. `create_A342()` Method

If an A342 sensor is connected to the Raspberry Pi, this method creates a `logger.measure.A342` object to control the sensor.

To change the behavior of data reading and writing for the A342 sensor, define an extended `A342` class and modify this method to return the object of that class.

6.2.3.3. `create_A352()` Method

If an A352 sensor is connected to the Raspberry Pi, this method creates a `logger.measure.A352` object to control the sensor.

To change the behavior of data reading and writing for the A352 sensor, define an extended `A352` class and modify this method to return the object of that class.

6.2.3.4. `create_reader_process()` Method

This method creates the `Reader` process used when a `Sensor` object starts measurement. Specifically, it creates the Reader process by specifying the `logger.measure.reader_job` function in `logger.core.LoggerProcess`.

If you want to customize the function executed in the `Reader` process, define an extended job function, and modify this method to use it.

6.2.3.5. `create_writer_process()` Method

This method creates the `Writer` process used when a `Sensor` object starts measurement. Specifically, it creates the Writer process by specifying the `logger.measure.writer_job` function in `logger.core.LoggerProcess`.

If you want to customize the function executed in the Writer process, define an extended job function, and modify this method to use it.

6.3. Customization Points for the VC-Based Vibration Measurement System

This application utilizes the extension mechanisms of Raspi Logger described above to enable vibration measurement using the VC Evaluation Library. The following customizations have been implemented:

6.3.1. `vc_calc_app.VcConfig` Class

This class inherits from `logger.core.Config` and serves as a data class that holds additional configuration parameters specific to this application.

6.3.2. `vc_calc_app.VcConfigurator` Class

This class inherits from `logger.core.Configurator`. It reads the additional configuration parameters from the configuration file and returns a `VcConfig` object.

6.3.3. `vc_calc_app.VcWriterArgs` Class

This class inherits from `logger.measure.WriterArgs`.
It is extended to perform calculations using the VC Evaluation Library as part of the output processing for measurement data.

The `WriterArgs` object is generated by the `Sensor` class for each sensor and holds sensor-specific parameters to support output processing. By extending this object, VC evaluation is integrated into the output process.

6.3.4. `vc_calc_app.VcGPIO` Class

This class is created to handle GPIO control based on VC evaluation results. The `VcWriterArgs` object uses it during processing to control GPIO.

6.3.5. `vc_calc_app.VcA352` Class

This class inherits from `logger.measure.A352`. It generates a `VcWriterArgs` object to perform VC evaluation as part of the output process.

To achieve this, the following method is extended:

6.3.5.1. `to_writer_args()` Method

This method is defined in the `logger.measure.Sensor` class and is responsible for generating a `WriterArgs` object. In `VcA352`, it returns a `VcWriterArgs` object.

6.3.6. `vc_calc_app.VcLoggerFactory` Class

This class inherits from `logger.core.LoggerFactoryImpl` and defines the following methods:

6.3.6.1. `create_configurator()` Method

Creates and returns a `VcConfigurator` object.

6.3.6.2. `create_A352()` Method

Creates and returns a `VcA352` object.

6.3.6.3. `create_A342()` Method

Since A342-series sensors are not supported by this application, this method raises an exception to indicate that the sensor is not supported when detected.

6.3.7. `vc_calc_app.__main__` Module

The `src/vc_calc_app/__main__.py` file defines the `__main__` module for the `vc_calc_app` package. It creates a `VcLoggerFactory` and passes it to the `logger.main` function.

```
import sys
from logger import main
from .factory import VcLoggerFactory

if __name__ == "__main__":
    code = main(factory=VcLoggerFactory())
    sys.exit(code)
```

Figure 6-3 `__main__` module

7. Contact Information

Seiko Epson Corporation

Sales Headquarters MD Sales Department

Contact Information via the Internet

https://www.epsondevice.com/sensing/en/privacy/area-select-inquiry-contact.html