Welcome to WMS Real-Time Data Python Library's Documentation

Contents:

- MEASUREMENT_RECORD_ON_100HZ
- MEASUREMENT_RECORD_ON_50HZ
- MEASUREMENT_RECORD_ON_20HZ
- MEASUREMENT_RECORD_ON_10HZ
- MEASUREMENT_RECORD_OFF
- MEASUREMENT_RECORD_OFF_200HZ
- MEASUREMENT_RECORD_OFF_100HZ
- MEASUREMENT_RECORD_OFF_50HZ
- MEASUREMENT_RECORD_OFF_20HZ
- MEASUREMENT_RECORD_OFF_10HZ
- BATTERY_CONDITION
- get_normalized_bit()
- CallbackType
  - CallbackType.ACK_RESPONSE
  - CallbackType.SENSOR_INFO
  - CallbackType.SENSOR_DATA
- CAckPacket
  - CAckPacket.__init__()
  - CAckPacket.Analyze()
  - CAckPacket.isCommandComplete()
  - CAckPacket.isEqualResponseCode()
  - CAckPacket.GetAckStatus()
  - CAckPacket.GetAckStatusStr()
  - CAckPacket.DebugPrint()
- CDataPacketCommon
  - CDataPacketCommon.__init__()
  - CDataPacketCommon.AnalyzeHeader()
  - CDataPacketCommon.GetProductID()
  - CDataPacketCommon.DebugPrint()
- CDataPacket_GetStatusInfo
  - CDataPacket_GetStatusInfo.__init__()
  - CDataPacket_GetStatusInfo.Analyze()
  - CDataPacket_GetStatusInfo.GetResultStr()
  - CDataPacket_GetStatusInfo.DebugPrint()
- CDataPacket_StartMeasure
  - CDataPacket_StartMeasure.__init__()
  - CDataPacket_StartMeasure.Analyze()
  - CDataPacket_StartMeasure.GetResultStr()
  - CDataPacket_StartMeasure.DebugPrint()

- CDataPacket_EndMeasure
  - CDataPacket_EndMeasure.__init__()
  - CDataPacket_EndMeasure.Analyze()
  - CDataPacket_EndMeasure.GetResultStr()
  - CDataPacket_EndMeasure.DebugPrint()
- CDataPacket
  - CDataPacket.__init__()
  - CDataPacket.SetupCallbackFunc()
  - CDataPacket.SetSensorList()
  - CDataPacket.CalcBCC()
  - CDataPacket.DoConvertSetup()
  - CDataPacket.DoConvert()
  - CDataPacket.DoConvertChannel()
  - CDataPacket.WaitAck()
  - CDataPacket.GetMeasurementHz()
  - CDataPacket.GetSendCommand()
  - CDataPacket.SendPacketASync()
  - CDataPacket.SendStartMeasureCommand()
  - CDataPacket.SendStopMeasureCommand()
  - CDataPacket.GetSendCommand_GetStatusInfo()
  - CDataPacket.GetSendCommand_SetSamplingFreq()
  - CDataPacket.GetSendCommand_SetMeasureTime()
  - CDataPacket.GetSendCommand_SetupMeasure()
  - CDataPacket.GetSendCommand_StartMeasure()
  - CDataPacket.GetSendCommand_StopMeasure()
  - CDataPacket.StartEventHandler()
  - CDataPacket.StopEventHandler()
  - CDataPacket.ReceivePacketASync()
  - CDataPacket.CheckHeader()
  - CDataPacket.CheckFooter()
  - CDataPacket.CheckBCC()
  - CDataPacket.AnalyzePacketThread()

LibSensor

- DEF_PRODUCT_ID_WirelessMotionSensorV2
- ACC_STRING
- ACC_VALUE
- GYRO_STRING
- GYRO_VALUE
- GetProductString()
- GetProductAccString()
- GetProductGyroString()

        - `GetSensorObject()`
        - `IsV2Sensor()`
        - `Sensor`
            - `Sensor.__init__()`
            - `Sensor.SetupADConvertParameters()`
            - `Sensor.SetACCGyroIndex()`
            - `Sensor.CalcAcc()`
            - `Sensor.CalcGyro()`
            - `Sensor.find_hosei_table_id()`
            - `Sensor.LoadHoseiFile()`
    - LibSettings
        - `DEFAULT_SETTINGS`
        - `CSettings`
            - `CSettings.__init__()`
            - `CSettings.Load()`
            - `CSettings.Validate()`
            - `CSettings.HasFutteMe()`

ライブラリの使用

WMSリアルタイム受信Pythonライブラリ
    |
    |- **build** => *実行可能ファイルにコンパイルされたサンプルテストアプリ*
    |- **html** => *APIドキュメントファイル*
    LibPacket.py => *パケット送受信とコマンド生成の処理、および解析のAPIクラス*
    LibSerial.py => *シリアルポート接続を使用するためのAPIクラス*
    LibSensor.py => *センサ情報の取り扱いのAPIクラス*
    LibSettings.py => *Json形式のセンサ情報設定ファイルのAPIクラス*
    SampleApp.py => *メインのサンプルテストアプリ(このPythonスクリプトを実行すると、サンプルアプリを起動できます。)*
    SampleGui.py => *メインサンプルテストアプリのGUIソース*
    GenerateEnv.bat => *このバッチファイルをWindows OSで実行して、必要なPythonパッケージをローカルのVenv環境にインストールします*
    GenerateEnv.sh => *このシェルスクリプトファイルをLinux OSで実行して、必要なPythonパッケージをローカルのVenv環境にインストールします。*
    requirements.txt => *必要なPythonパッケージのリスト(インストール用)*
    settings.json => *センサID、製品ID、および補正ファイル名のリストの設定ファイル*

必要条件

対応センサ：

- 小型9軸ワイヤレスモーションセンサV2

- 薄型9軸ワイヤレスモーションセンサV2

対応OS :

- Windows 10/11
- CentOS 7/8
- Ubuntu 20.04

Pythonバージョン : Python3.9以降(Windows)、Python3.6以上(Linux)
必要なPythonパッケージ

- pyserial
- sphinx
- sphinx-rtd-theme

Windows環境でGenerateEnv.bat、 Linux環境でGenerateEnv.shを実行して、Venvローカル環境を自動生成し、必要なPythonパッケージをインストールすることができます。

GUI Tkinterパッケージが必要
現在のPythonにTkinterがインストールされているか確認するには、
> python -m tkinter

Tkinter GUI ウィンドウが表示されている場合はインストールされています。表示されていない場合は、次のコマンドを使用してインストールします。
(CentOS 7/8場合)
> sudo yum -y install python3-tkinter
(Ubuntu 20.04場合)
> sudo apt install python3-tk

シリアルポート設定 (Linux環境)

シリアルポートにアクセスするには、現在のユーザーにアクセス許可を設定する必要があります。そうしないと、アクセスするために常にrootユーザーが必要になります。
※Windows環境では以下の必要はありません。

【例】現在のユーザーアカウントからシリアルポート /dev/ttyACM0 にアクセスするには、

以下のようにシリアルポートのグループ名を確認します。
> sudo ls -al /dev/ttyACM0
crw-rw---- 1 root dialout 166, 0 9月 4 19:37 /dev/ttyACM0

次に、同じdialoutグループを現在のユーザーアカウントに割り当てます。
> sudo usermod -aG dialout $USER

設定後、現在のユーザーはルート認証なしでシリアルポートにアクセスできるようになります。

サンプルスクリプトの実行方法

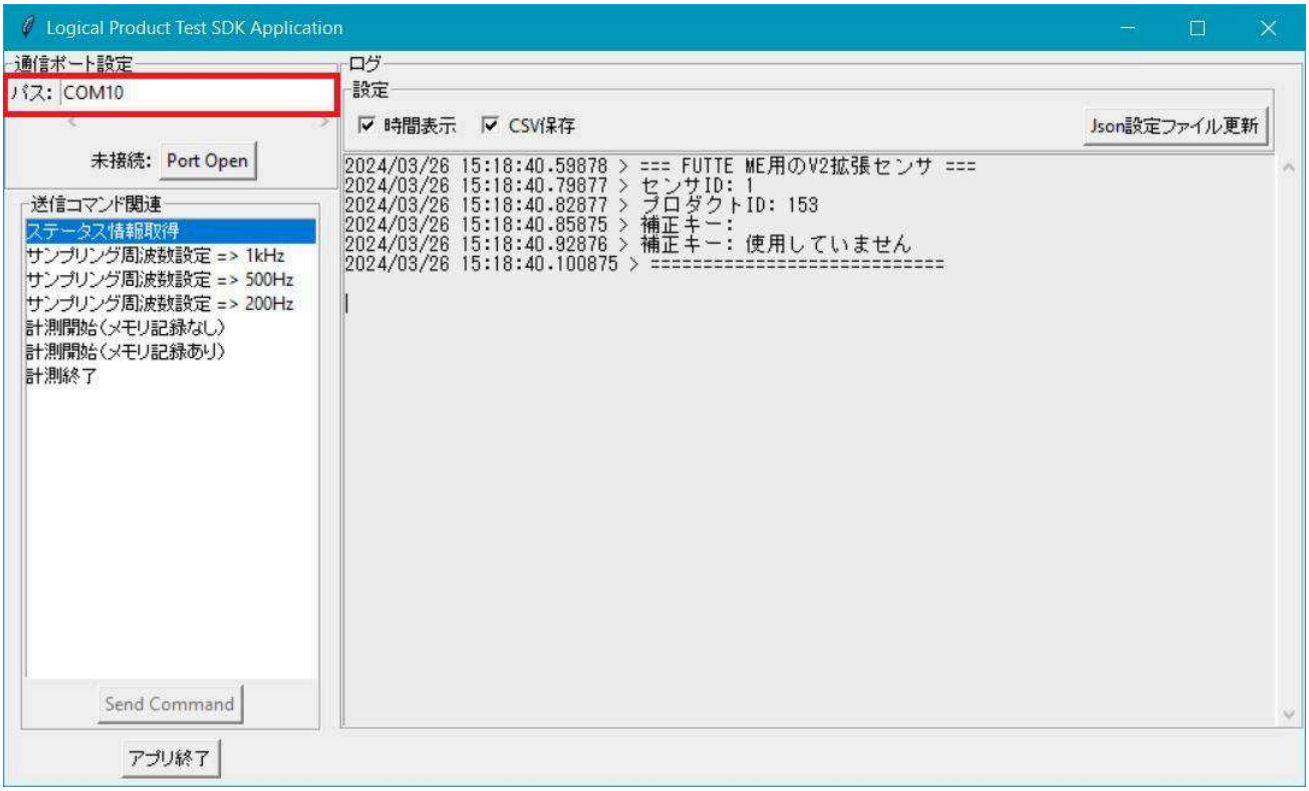
SampleApp.pyを実行する前に、次のコマンドを入力してローカルのPython環境をアクティブにします。
- (Windows環境 - PowerShell) > ./.venv/Scripts/Activate
- (Linux環境 - Terminal) > source ./.venv/bin/activate

このサンプルスクリプト(SampleApp.py)を試すには、ターミナルに次のコマンドを入力します。
> python SampleApp.py

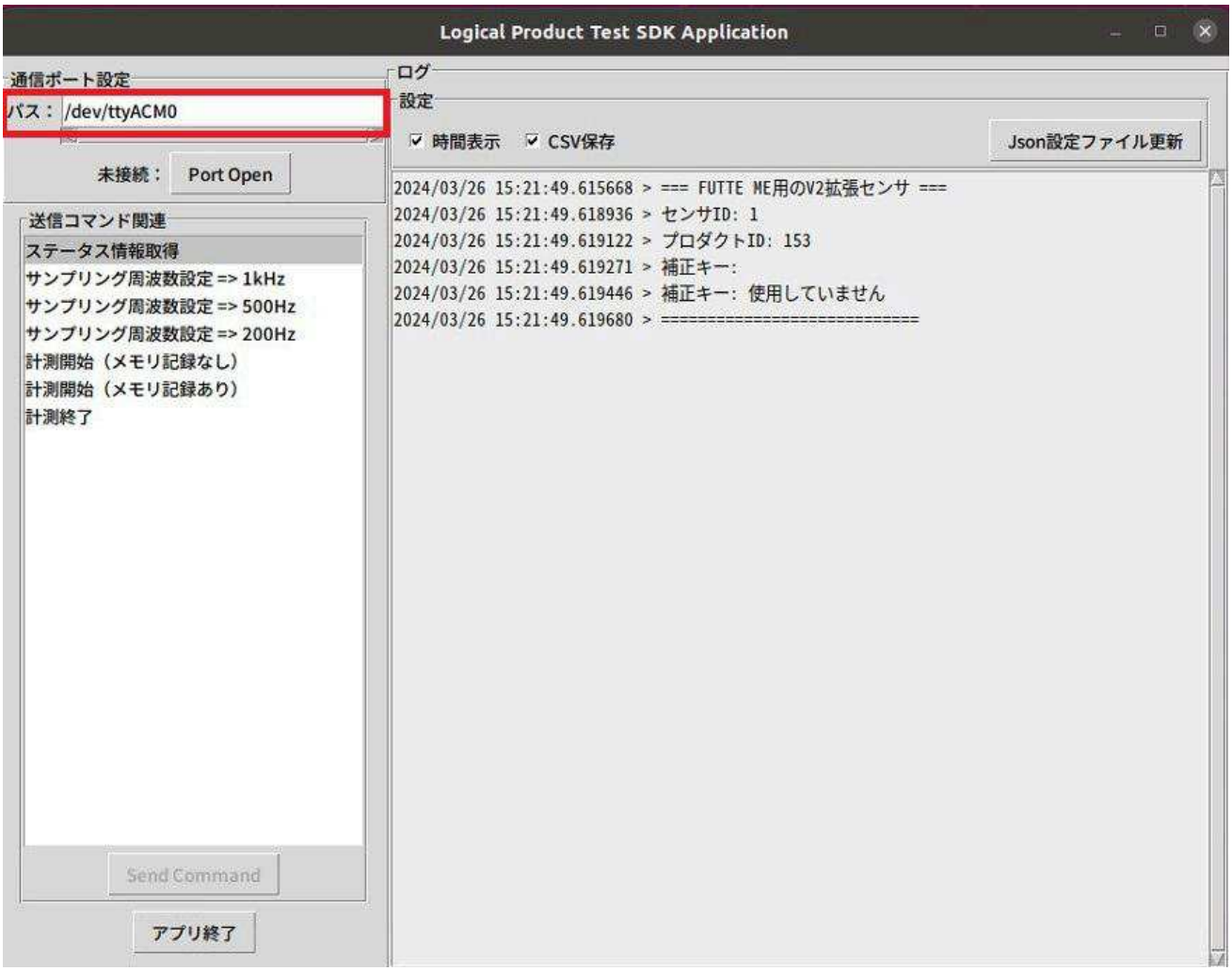
上記のスクリプトを実行すると、利用可能なシリアル ポートのリストが表示されます。 次のように適切な値を入力するか、空白のままにして Enter キーを押してください。

(Windows環境)
> COM10 - USB Serial Device (COM10)
> Select Port: COM
(Linux環境)
> /dev/ttyACM0 - LP-RF24DS-03
> Select Port:

Windows環境の場合は、上記サンプルと同様に COM 番号を 10 などに入力します。



Linux環境の場合は、上記サンプルと同様に /dev/ttyACM0 などのポート名を入力します。



[Port Open]ボタンをクリックすると、センサが接続されているシリアルポートが開きます。 シリアルポートがオープンされると、送信コマンド関連テーブルで [Send Command] ボタンをさまざまなコマンドで使用できるようになります。

スクリプトアプリを終了する場合は、[アプリ終了]ボタンをクリックしてください。

設定ファイル (settings.json)

settings.json ファイルには、センサID、製品ID、およびその補正ファイルに関する情報が含まれています。 センサ計測前に使用します。 このファイルは、Python Script API ライブラリと同じディレクトリにあります。 存在しない場合は、デフォルト設定を使用して自動的に作成されます。

Key	Value
id	センサID [数値]
pid	製品ID [数値]
Key	補正ファイル名 [文字列]

製品ID	Value
80	• 小型9軸ワイヤレスモーションセンサV2 • 薄型9軸ワイヤレスモーションセンサV2

settings.json ファイルの設定を追加または変更するには、このファイルをテキストエディターで開きます。 編集・保存後、[Json設定ファイル更新]ボタンをクリックすると、次回センサーを再度測定する際に正しいセンサー情報と補正データが使用されます。

【例】センサが設定アプリで補正された場合、補正されたファイルはこのjson設定ファイルに登録される必要があります。
- 補正後に補正ファイル (例: 6201451F5B6343A1853387A39D0E6BD7.wsc) が生成されます
- settings.jsonファイルを開く
- 次のようにjsonファイルに補正ファイル名を追加
- "key": "" => "key": "6201451F5B6343A1853387A39D0E6BD7.wsc"
- settings.jsonファイルを保存

※このjsonファイルは使用するセンサの総数を管理しています。 センサの数を増減する場合は、それに応じてこのJsonファイルを必ず編集してください。
※補正キーファイルを使用しない場合は、補正ファイル名を空文字列のままにしてください。
【例】"key": ""

Python Script APIsマニュアルの作り方

このマニュアルを作成する前に、ローカルのPython環境にアクティブことを確認してください。

ターミナルで、ディレクトリを[docs]に変更します。
> cd docs

次のコマンドを実行してマニュアルをビルドします。
- (Windows環境 - PowerShell) > ./make html
- (Linux環境 - Terminal) > make html

ビルドが成功すると、HTMLページが_build/htmlディレクトリに作成されます。

スタンドアロンPythonアプリの作り方

pyinstallerをダウンロードして、ローカルのPython環境にインストールします。
> pip install pyinstaller

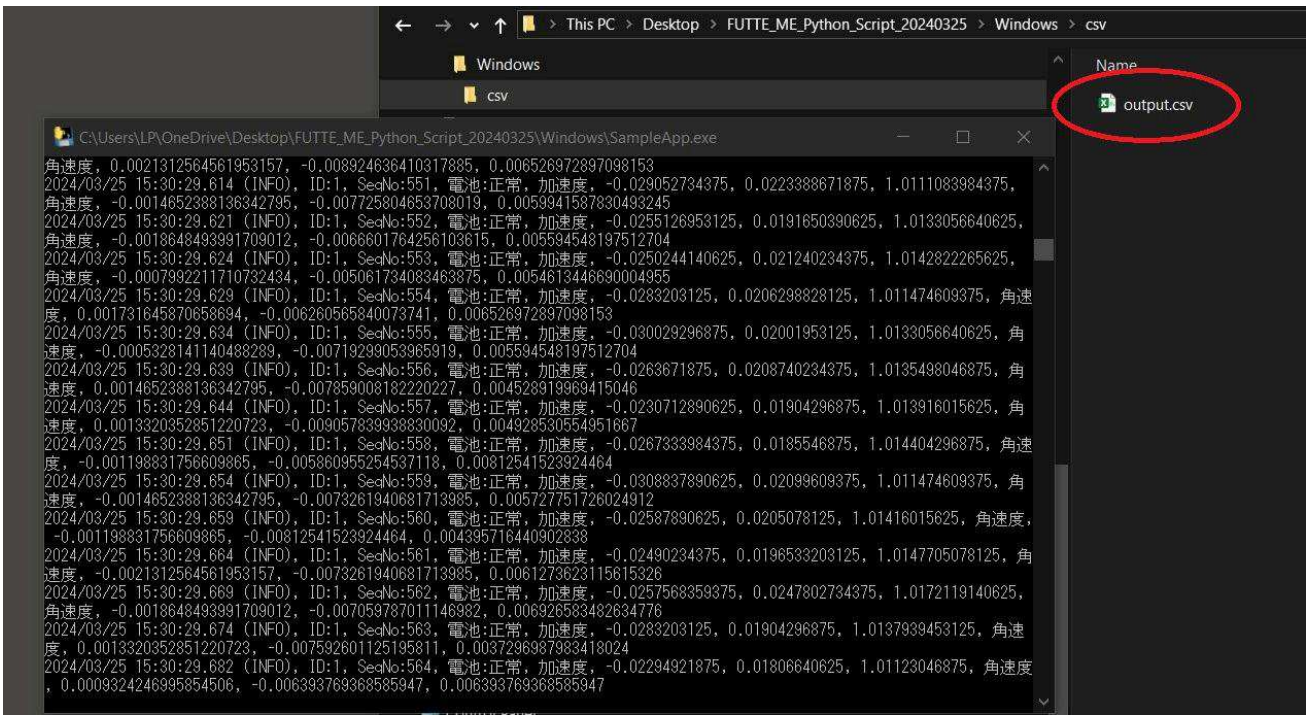
Pythonスクリプトのソースファイルディレクトリからスタンドアロンアプリをビルドします。
> python -m PyInstaller SampleApp.py --onefile

ビルドが成功すると、[dist]ディレクトリが作成され、ここにスタンドアロンPythonアプリが生成されます。

※アプリを適切に実行するには、settings.jsonファイルと必要なすべての調整ファイル(.wsc) をこのアプリの同じディレクトリにコピーしてください。

Windows OSでアプリを実行するには、アプリ [SampleApp.exe] をダブルクリックするだけです。 Linux OSでは、ターミナルを開いてディレクトリをアプリの場所に変更し、「./SampleApp」と入力して実行します。

センサが測定を開始すると、センサデータはコンソールウィンドウと[csv]ディレクトリにあるCSVファイル[output.csv]の両方に出力されます。センサデータのCSVファイルへの保存をオフにするには、[CSV保存]チェックボックスをクリックして保存を停止します。 もう一度チェックを入れて保存をオンにします。



【例】サンプル

```
import LibSerial
import LibPacket
import LibSensor
```

```
import LibSettings

#確認応答を受信のコールバック関数
#パラメーター [id]: センサIDを返す
#パラメーター [ackPacket]: LibPacket.CAckPacket クラスオブジェクトを使用して、確認情報にアクセス
def ReceiveAckCallback(id, ackPacket) -> None:
    ...

#センサーステータス情報を受信のコールバック関数
#パラメーター [id]: センサIDを返す
#パラメーター [infoPacket]: LibPacket.CDataPacket_GetStatusInfo クラスオブジェクトを使用して、センサのステータス情報にアクセス
def ReceiveInfoCallback(id, infoPacket) -> None:
    ...

#センサーデータを受信のコールバック関数
#パラメーター [id]: センサIDを返す
#パラメーター [sequenceNo]: 現在のデータパッケージのシーケンス番号を返す
#パラメーター [convertedData]: ADC 生データから変換されたデータを返す
def ReceiveDataCallback(id, sequenceNo, convertedData) -> None:
    ...

#Json 設定ファイルを読み込み、センサオブジェクトのリストを作成
settings = LibSettings.CSettings()

#シリアルポートオブジェクトを作成
SerialPort = LibSerial.CSerialPort()

#データパケットオブジェクトを作成
DataPacket = LibPacket.CDataPacket(self.SerialPort)

#コールバック関数を作成
DataPacket.SetupCallbackFunc(ReceiveAckCallback,
LibPacket.CallbackType.ACK_RESPONSE)
DataPacket.SetupCallbackFunc(ReceiveInfoCallback, LibPacket.CallbackType.SENSOR_INFO)
DataPacket.SetupCallbackFunc(ReceiveDataCallback,
LibPacket.CallbackType.SENSOR_DATA)

#センサーリストを設定
DataPacket.SetSensorList(settings.sensor_list)

#シリアルポート開く処理を開始
SerialPort.OpenPort(PortName)
```

#受信スレッド開始
DataPacket.StartEventHandler()

#すべてのセンサのステータスを取得 (センサ情報を更新)
DataPacket.GetSendCommand(LibPacket.DEF_SENDCOMMAND_ID_GETSTATUSINFO, 0, 0, 0)

#計測開始コマンドを送信 (1台、メモリ記録無し、1kHz、4時間計測時間)
DataPacket.GetSendCommand(LibPacket.DEF_SENDCOMMAND_ID_STARTMEASURE, 1, 1000, 1800)

... センサーデータ受信 ...
... ReceiveDataCallbackはデータを受信
... ReceiveDataCallbackはデータを受信
... ReceiveDataCallbackはデータを受信

#受信スレッド停止
DataPacket.StopEventHandler()

#シリアルポート閉じる処理
SerialPort.ClosePort()

最終更新日: 2025-11-27