山梨職業能力開発促進センター

クラウド活用による IoTシステム構築技術

生産現場におけるクラウドサービスを利用したアプリ開発

模範解答

＜目次＞

第１章　ソケット通信　模範解答

第２章　ＳＮＳを利用したＩｏＴアプリケーション開発　模範解答

第３章　ライブラリモジュール

```python
import socket
import time
from tmp102 import TMP102


"""

  練習問題　(サーバへのセンサデータ送信)

  ファイル名：tmpAppli.py
        内容：オンボード温度センサ(TMP102)から温度データを取得し，サーバへ送信する
                サーバから同じデータが送り返されてきたデータを受信してコンソール表示する
"""

#TMP102インスタンス生成
tmp = TMP102()



while True:

    #温度データ取得
    t = tmp.readTemperature()

    #ソケット生成
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
        #サーバ接続
        client.connect(("10.0.3.131",8000))

        #データ送信 (send)
        client.send(f"{t:.02f}[℃]".encode())

        #データ受信(recv)
        data = client.recv(128)

        #表示
        print(data.decode())

        #回線切断要求
        client.close()

    time.sleep(1.0)
```

```python
#-*- coding:utf-8 -*-
import socket
import sys

"""

  練習問題　(搬送負荷装置における稼働状況の遠隔監視)

  ファイル名：machine_state.py
        内容：搬送負荷装置の稼働状況を取得し、コンソールへ表示する
"""
#ソケット生成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:

    #サーバ接続 (connect)
    client.connect(('10.0.11.220',5001));

    #---------------------------------
    #要求伝文 (送信パケット) セット
    # M200-M202まで3ビット分を取得
    #---------------------------------
    cmd = "00FF00004D20000000C80300"

    #データ送信(send)
    client.send(cmd.encode())

    #データ受信(recv)
    Data = client.recv(128).decode()

    #接点(M200－M202)情報の取得(str型からint型へ変換する)
    M200_state = int(Data[4])
    M201_state = int(Data[5])
    M202_state = int(Data[6])

    if M202_state == 1:
        print("システム緊急停止中です")
        sys.exit()

    if M200_state == 1:
        print("システム停止中です")
        sys.exit()

    if M201_state == 1:
        print("システム稼働中です")
        sys.exit()
```

```python
#-*- coding:utf-8 -*-
import socket
import sys

"""

  練習問題　(搬送負荷装置における生産管理の遠隔監視)

  ファイル名：manufacture_state.py
        内容：搬送負荷装置の製造状況を取得し、コンソールへ表示する
"""

#ソケット生成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:

    #サーバ接続 (connect)
    client.connect(('10.0.11.220',5001));

    #---------------------------------
    #要求伝文 (送信パケット) セット
    # D200-D203まで4ワード分を取得
    #---------------------------------
    cmd = "01FF000044200000000C80400"

    #データ送信(send)
    client.send(cmd.encode())

    #データ受信(recv)
    Data = client.recv(128).decode()

    #各値を取得
    D200_value = int(Data[4:8], 16)   # 4bit-7bit : 生産目標値
    D201_value = int(Data[8:12], 16)  # 8bit-11bit: 生産個数
    D202_value = int(Data[12:16], 16) #12bit-15bit: 良品数
    D203_value = int(Data[16:20], 16) #16bit-19bit: 不良品数

    #取得した値を表示
    print(f"生産目標数:{D200_value}")
    print(f"生産数:{D201_value}")
    print(f"良品数:{D202_value}")
    print(f"不良品数:{D203_value}")

```

```python
#-*- coding:utf-8 -*-
import socket
import sys

"""

  練習問題　(搬送負荷装置における生産目標数の設定)

  ファイル名：manufacture_state.py
        内容：コンソールから搬送負荷装置の生産数を入力し、
              PLCへ遠隔設定をする。
"""

#ソケット生成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:

    #サーバ接続 (connect)
    client.connect(('10.0.11.220',5001));

    #コンソールから生産目標値を入力
    pt = int(input("生産目標数をセットしてください>"))

    #---------------------------------
    #要求伝文 (送信パケット) セット
    # D300をセット
    #---------------------------------
    senddata = f"03FF0000442000000012C0100{pt:04X}"  #16進数に変換し文字列にセット

    #データ送信(send)
    client.sendall(senddata.encode())

    #データ受信(recv)
    Data = client.recv(128).decode()



    #---------------------------------
    #要求伝文 (送信パケット) セット
    # D200を取得
    #---------------------------------
    cmd = "01FF000044200000000C80100"

    #データ送信(send)
    client.send(cmd.encode())

    #データ受信(recv)
    Data = client.recv(128).decode()

    #生産目標数を取得
    D200_value = int(Data[4:8], 16)

    #生産目標数を表示
    print(f"生産目標数:{D200_value}")

```

```
"""

サンプルコード
 (LINEからのメッセージエコーアプリ)

 Ver:1.0
Date:2022/02/09
Note:LINEからのメッセージエコーアプリ
File:messageEvent.py
"""
#-----------------------------------------------
#システム標準関連ライブラリ
#-----------------------------------------------
from __future__ import unicode_literals
import errno
import os
import sys
import subprocess
import tempfile

#コマンドライン引数関連のライブラリ
from argparse import ArgumentParser

#-----------------------------------------------
#Webフレームワーク関連ライブラリ
#-----------------------------------------------
#flask関連ライブラリ
from flask import Flask, request, abort
#flaskインスタンス生成
app = Flask(__name__)

#-----------------------------------------------
#LINE関連ライブラリ
#-----------------------------------------------
#LINE API
from linebot import (
    LineBotApi, WebhookHandler
)

#LINE署名不一致による例外イベントの定義
from linebot.exceptions import (
    InvalidSignatureError
)

#LINEイベント
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    SourceUser, SourceGroup, SourceRoom,
    TemplateSendMessage, ConfirmTemplate, MessageTemplateAction,
    ButtonsTemplate, ImageCarouselTemplate, ImageCarouselColumn,
    URITemplateAction,
    PostbackTemplateAction, DatetimePickerTemplateAction,
    CarouselTemplate, CarouselColumn, PostbackEvent,
    StickerMessage, StickerSendMessage, LocationMessage, LocationSendMessage,
    ImageMessage, VideoMessage, AudioMessage, FileMessage,
    UnfollowEvent, FollowEvent, JoinEvent, LeaveEvent, BeaconEvent
)

```

```
59
60
61 #----------------------------------------------
62 #環境変数の読み込み
63 #----------------------------------------------
64 #LINEチャネルシークレットトークン環境変数読み込み
65 channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
66 #LINEチャネルアクセストークン環境変数読み込み
67 channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
68 #もしチャネルシークレットトークンが読み込みできなければプロセス停止
69 if channel_secret is None:
70     print('Specify LINE_CHANNEL_SECRET as environment variable.')
71     sys.exit(1)
72 #もしチャネルアクセストークンが読み込みできなければプロセス停止
73 if channel_access_token is None:
74     print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
75     sys.exit(1)
76
77 #----------------------------------------------
78 #LINE APIインスタンス生成
79 #----------------------------------------------
80 #LINEbotAPIインスタンス生成
81 line_bot_api = LineBotApi(channel_access_token)
82 #WebhookHandlerインスタンス生成
83 handler = WebhookHandler(channel_secret)
84
85 #----------------------------------------------
86 #ダウンロードファイルの保存先ディレクトリ
87 #----------------------------------------------
88 #ファイルパスを生成
89 static_tmp_path = os.path.join(os.path.dirname(__file__), 'static', 'tmp')
90
91 #ディレクトリ作成関数
92 def make_static_tmp_dir():
93     try:
94         #ディレクトリ作成
95         os.makedirs(static_tmp_path)
96
97     #例外処理
98     except OSError as exc:
99         #もしファイルもしくはディレクトリが存在していたら何もしない
100         if exc.errno == errno.EEXIST and os.path.isdir(static_tmp_path):
101             pass
102         #もし別要因によるものなら例外を通知する
103         else:
104             raise
105
106
107 #----------------------------------------------
108 #電子署名検証
109 #(LINEbotからの呼び出しがHTTPSのため必須)
110 #----------------------------------------------
111 @app.route("/callback",methods=['POST'])
112 def callback():
113     # HTTPリクエストのヘッダを取得
114     signature = request.headers['X-Line-Signature']
115
116     # HTTPリクエスト(POST)のボディを取得
```

```
117     body = request.get_data(as_text=True)
118     app.logger.info("Request body: " + body)
119 
120     # 署名を検証し、問題なければ@handler.addに定義している関数を呼び出す
121     try:
122         handler.handle(body, signature)
123     except InvalidSignatureError:
124         abort( 400 )
125 
126     return 'OK'
127 
128 #----------------------------------------------
129 #メッセージイベントメソッド
130 #----------------------------------------------
131 @handler.add(MessageEvent, message=TextMessage)
132 def handle_text_message(event):
133     #LINEから受信した文字を代入
134     text = event.message.text
135     print(text)
136     #同じ文字をLINEで送り返す
137     line_bot_api.reply_message(
138         event.reply_token,          #リクエスト応答用トークン
139         TextSendMessage(text)   #メッセージ送信
140     )
141 
142 if __name__== "__main__":
143     #ダウンロード先ディレクトリの生成
144     make_static_tmp_dir()
145 
146     #Flask起動
147     app.run(debug=True, port=8080)
148 
```

```
"""

練習問題　（システム内部の温度取得）

 Ver:1.0
Date:2022/02/17
Note:温度センサ（TMP102）からの情報をLINEでメッセージ送信する
File:tempEvent.py

"""
#----------------------------------------------
#システム標準関連ライブラリ
#----------------------------------------------
from __future__ import unicode_literals
import errno
import os
import sys
import subprocess
import tempfile

from tmp102 import TMP102

#コマンドライン引数関連のライブラリ
from argparse import ArgumentParser

#----------------------------------------------
#Webフレームワーク関連ライブラリ
#----------------------------------------------
#flask関連ライブラリ
from flask import Flask, request, abort
#flaskインスタンス生成
app = Flask(__name__)

#----------------------------------------------
#LINE関連ライブラリ
#----------------------------------------------
#LINE API
from linebot import (
    LineBotApi, WebhookHandler
)

#LINE署名不一致による例外イベントの定義
from linebot.exceptions import (
    InvalidSignatureError
)

#LINEイベント
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    SourceUser, SourceGroup, SourceRoom,
    TemplateSendMessage, ConfirmTemplate, MessageTemplateAction,
    ButtonsTemplate, ImageCarouselTemplate, ImageCarouselColumn,
    URITemplateAction,
    PostbackTemplateAction, DatetimePickerTemplateAction,
    CarouselTemplate, CarouselColumn, PostbackEvent,
    StickerMessage, StickerSendMessage, LocationMessage, LocationSendMessage,
    ImageMessage, VideoMessage, AudioMessage, FileMessage,
    UnfollowEvent, FollowEvent, JoinEvent, LeaveEvent, BeaconEvent
```

```
)



#---------------------------------------------
#環境変数の読み込み
#---------------------------------------------
#LINEチャネルシークレットトークン環境変数読み込み
channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
#LINEチャネルアクセストークン環境変数読み込み
channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
#もしチャネルシークレットトークンが読み込みできなければプロセス停止
if channel_secret is None:
    print('Specify LINE_CHANNEL_SECRET as environment variable.')
    sys.exit(1)
#もしチャネルアクセストークンが読み込みできなければプロセス停止
if channel_access_token is None:
    print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
    sys.exit(1)

#---------------------------------------------
#LINE APIインスタンス生成
#---------------------------------------------
#LINEbotAPIインスタンス生成
line_bot_api = LineBotApi(channel_access_token)
#WebhookHandlerインスタンス生成
handler = WebhookHandler(channel_secret)

#---------------------------------------------
#ダウンロードファイルの保存先ディレクトリ
#---------------------------------------------
#ファイルパスを生成
static_tmp_path = os.path.join(os.path.dirname(__file__), 'static', 'tmp')

#ディレクトリ作成関数
def make_static_tmp_dir():
    try:
        #ディレクトリ作成
        os.makedirs(static_tmp_path)

    #例外処理
    except OSError as exc:
        #もしファイルもしくはディレクトリが存在していたら何もしない
        if exc.errno == errno.EEXIST and os.path.isdir(static_tmp_path):
            pass
        #もし別要因によるものなら例外を通知する
        else:
            raise


#---------------------------------------------
#電子署名検証
#(LINEbotからの呼び出しがHTTPSのため必須)
#---------------------------------------------
@app.route("/callback",methods=['POST'])
def callback():
    # HTTPリクエストのヘッダを取得
    signature = request.headers['X-Line-Signature']
```

```
117
118     # HTTPリクエスト(POST)のボディを取得
119     body = request.get_data(as_text=True)
120     app.logger.info("Request body: " + body)
121
122     # 署名を検証し、問題なければ@handler.addに定義している関数を呼び出す
123     try:
124         handler.handle(body, signature)
125     except InvalidSignatureError:
126         abort( 400 )
127
128     return 'OK'
129
130 #----------------------------------------------
131 #メッセージイベントメソッド
132 #----------------------------------------------
133 @handler.add(MessageEvent, message=TextMessage)
134 def handle_text_message(event):
135     #LINEから受信した文字を代入
136     text = event.message.text
137
138     #もし受信メッセージが「温度を表示」だったら…
139     if text=="温度を表示":
140         #温度を取得
141         tmp = TMP102()
142         t = tmp.readTemperature()
143
144         #温度をLINEで送り返す
145         line_bot_api.reply_message(
146             event.reply_token,          #リクエスト応答用トークン
147             TextSendMessage(text=f"現在 {t:.1f}℃です") #メッセージ送信
148         )
149
150     else:
151         #「わかりません」をLINEで送り返す
152         line_bot_api.reply_message(
153             event.reply_token,          #リクエスト応答用トークン
154             TextSendMessage("わかりません") #メッセージ送信
155         )
156 if __name__== "__main__":
157     #ダウンロード先ディレクトリの生成
158     make_static_tmp_dir()
159
160     #Flask起動
161     app.run(debug=True, port=8080)
162
```

```
"""

練習問題　(搬送負荷装置における稼働状況の問い合わせ)

 Ver:1.0
Date:2022/02/17
Note:搬送負荷装置の稼働状況をLINEでメッセージ送信する
File:machineStateEvent.py

"""
#----------------------------------------------
#システム標準関連ライブラリ
#----------------------------------------------
from __future__ import unicode_literals
import errno
import os
import sys
import subprocess
import tempfile
from PLCCOM import PLCCOM

#コマンドライン引数関連のライブラリ
from argparse import ArgumentParser

#----------------------------------------------
#Webフレームワーク関連ライブラリ
#----------------------------------------------
#flask関連ライブラリ
from flask import Flask, request, abort
#flaskインスタンス生成
app = Flask(__name__)

#----------------------------------------------
#LINE関連ライブラリ
#----------------------------------------------
#LINE API
from linebot import (
    LineBotApi, WebhookHandler
)

#LINE署名不一致による例外イベントの定義
from linebot.exceptions import (
    InvalidSignatureError
)

#LINEイベント
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    SourceUser, SourceGroup, SourceRoom,
    TemplateSendMessage, ConfirmTemplate, MessageTemplateAction,
    ButtonsTemplate, ImageCarouselTemplate, ImageCarouselColumn,
    URITemplateAction,
    PostbackTemplateAction, DatetimePickerTemplateAction,
    CarouselTemplate, CarouselColumn, PostbackEvent,
    StickerMessage, StickerSendMessage, LocationMessage, LocationSendMessage,
    ImageMessage, VideoMessage, AudioMessage, FileMessage,
    UnfollowEvent, FollowEvent, JoinEvent, LeaveEvent, BeaconEvent
)
```

```
59
60 #----------------------------------------------
61 #環境変数の読み込み
62 #----------------------------------------------
63 #LINEチャネルシークレットトークン環境変数読み込み
64 channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
65 #LINEチャネルアクセストークン環境変数読み込み
66 channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
67 #もしチャネルシークレットトークンが読み込みできなければプロセス停止
68 if channel_secret is None:
69     print('Specify LINE_CHANNEL_SECRET as environment variable.')
70     sys.exit(1)
71 #もしチャネルアクセストークンが読み込みできなければプロセス停止
72 if channel_access_token is None:
73     print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
74     sys.exit(1)
75
76 #----------------------------------------------
77 #LINE APIインスタンス生成
78 #----------------------------------------------
79 #LINEbotAPIインスタンス生成
80 line_bot_api = LineBotApi(channel_access_token)
81 #WebhookHandlerインスタンス生成
82 handler = WebhookHandler(channel_secret)
83
84 #----------------------------------------------
85 #ダウンロードファイルの保存先ディレクトリ
86 #----------------------------------------------
87 #ファイルパスを生成
88 static_tmp_path = os.path.join(os.path.dirname(__file__), 'static', 'tmp')
89
90 #ディレクトリ作成関数
91 def make_static_tmp_dir():
92     try:
93         #ディレクトリ作成
94         os.makedirs(static_tmp_path)
95
96     #例外処理
97     except OSError as exc:
98         #もしファイルもしくはディレクトリが存在していたら何もしない
99         if exc.errno == errno.EEXIST and os.path.isdir(static_tmp_path):
100             pass
101         #もし別要因によるものなら例外を通知する
102         else:
103             raise
104
105 #----------------------------------------------
106 #電子署名検証
107 #(LINEbotからの呼び出しがHTTPSのため必須)
108 #----------------------------------------------
109 @app.route("/callback",methods=['POST'])
110 def callback():
111     # HTTPリクエストのヘッダを取得
112     signature = request.headers['X-Line-Signature']
113
114     # HTTPリクエスト(POST)のボディを取得
115     body = request.get_data(as_text=True)
116     app.logger.info("Request body: " + body)
```

```
117
118     # 署名を検証し、問題なければ@handler.addに定義している関数を呼び出す
119     try:
120         handler.handle(body, signature)
121     except InvalidSignatureError:
122         abort( 400 )
123     return 'OK'
124
125 #---------------------------------------------
126 #メッセージイベントメソッド
127 #---------------------------------------------
128 @handler.add(MessageEvent, message=TextMessage)
129 def handle_text_message(event):
130     #LINEから受信した文字を代入
131     text = event.message.text
132
133     if text=="稼働状況":
134         #機械の稼働状況を取得
135         plc=PLCCOM("10.0.11.220", 5001)
136         state = plc.read_machine_state()        #稼働状況
137         pdt = plc.read_product()                #生産数
138         gd_pdt = plc.read_good_product()        #良品数
139         er_pdt = plc.read_defective_product()   #不良品の数
140
141         #機械の稼働状況を判断、メッセージをセット
142         print(f"state:{state}")
143         if state=="STOP":
144             m_state = "現在停止中です¥r¥n"
145         elif state=="DRIVE":
146             m_state = "現在稼働中です¥r¥n"
147         elif state=="EMERGENCY":
148             m_state = "現在緊急停止中です¥r¥n"
149         else:
150             m_state = "システムエラーが発生しました"
151
152         message = m_state + ¥
153                   "---------------¥r¥n" + ¥
154                   f"生産数:{pdt}¥r¥n" + ¥
155                   f"良品数:{gd_pdt}¥r¥n" + ¥
156                   f"不良品:{er_pdt}"
157
158         #稼働状況をLINEで送り返す
159         line_bot_api.reply_message(
160             event.reply_token,          #リクエスト応答用トークン
161             TextSendMessage(text=message)   #メッセージ送信
162         )
163     else:
164         #「わかりません」をLINEで送り返す
165         line_bot_api.reply_message(
166             event.reply_token,          #リクエスト応答用トークン
167             TextSendMessage("わかりません") #メッセージ送信
168         )
169 if __name__== "__main__":
170     #ダウンロード先ディレクトリの生成
171     make_static_tmp_dir()
172     #Flask起動
173     app.run(debug=True, port=8080)
174
```

```
"""
サンプルコード
 (通知機能の実装)

 Ver:1.0
Date:2022/02/09
Note:通知機能の実装 (メッセージ)
File:pushMessage.py
"""
from linebot import LineBotApi
from linebot.models import TextSendMessage
import os
#----------------------------------------------
#環境変数の読み込み
#----------------------------------------------
#LINEチャネルシークレットトークン環境変数読み込み
channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
#LINEチャネルアクセストークン環境変数読み込み
channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
#もしチャネルシークレットトークンが読み込みできなければプロセス停止
if channel_secret is None:
    print('Specify LINE_CHANNEL_SECRET as environment variable.')
    sys.exit(1)
#もしチャネルアクセストークンが読み込みできなければプロセス停止
if channel_access_token is None:
    print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
    sys.exit(1)

#----------------------------------------------
#LINE APIインスタンス生成
#----------------------------------------------
line_bot_api = LineBotApi(channel_access_token)

#----------------------------------------------
#メイン関数
#----------------------------------------------
def main():
    #ユーザーＩＤ
    user_id = "U7484546b91ce74225ad04041c2e6bd7d"

    #メッセージ通知
    line_bot_api.push_message(
        user_id,
        messages=TextSendMessage(text="こんにちは！")
    )

if __name__=="__main__":
    main()


```

```
"""

練習問題　(通知機能を利用した非常停止の通知)

 Ver:1.0
Date:2022/02/17
Note:搬送負荷装置の非常停止時にLINEメッセージを通知する
File:EmergencyNotification.py

"""

from linebot import LineBotApi
from linebot.models import TextSendMessage
import os
import time
import datetime
from PLCCOM import PLCCOM



#----------------------------------------------
#環境変数の読み込み
#----------------------------------------------
#LINEチャネルシークレットトークン環境変数読み込み
channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
#LINEチャネルアクセストークン環境変数読み込み
channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
#もしチャネルシークレットトークンが読み込みできなければプロセス停止
if channel_secret is None:
    print('Specify LINE_CHANNEL_SECRET as environment variable.')
    sys.exit(1)
#もしチャネルアクセストークンが読み込みできなければプロセス停止
if channel_access_token is None:
    print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
    sys.exit(1)

#----------------------------------------------
#LINE APIインスタンス生成
#----------------------------------------------
line_bot_api = LineBotApi(channel_access_token)

#----------------------------------------------
#PLCCOMインスタンス生成
#----------------------------------------------
plc = PLCCOM("10.0.11.220",5001)


def main():
    #ユーザーID設定
    user_id = "U7484546b91ce74225ad04041c2e6bd7d"

    #設備の状態保存用変数
    old_state = "STOP"  #初期状態は停止状態

    while True:
        #稼働状況の取得
        new_state = plc.read_machine_state()

```

```
        if old_state!=new_state:    #もしold_stateとnew_stateが等しくなければ…
            if new_state=="EMERGENCY":  #もしnew_stateが"EMERGENCY"
                #現在の日時を取得
                days = datetime.datetime.now().strftime("%Y/%m/%d %H:%M")

                #LINE通知
                line_bot_api.push_message(
                    user_id,
                    messages=TextSendMessage(text=f"{days}¥r¥n設備が緊急停止しました")
                )


            old_state = new_state

        time.sleep(1.0)
if __name__=="__main__":
    main()

```

```
"""
サンプルコード
 （ボタンテンプレートを利用した搬送負荷装置の遠隔制御）

 Ver:1.0
Date:2022/02/17
Note:ボタンテンプレートを利用して搬送負荷装置の遠隔制御を行う
File:RemoteControl.py
"""
#-------------------------------------------
#システム標準関連ライブラリ
#-------------------------------------------
from __future__ import unicode_literals
import errno
import os
import sys
import subprocess
import tempfile

from PLCCOM import PLCCOM

#コマンドライン引数関連のライブラリ
from argparse import ArgumentParser

#-------------------------------------------
#Webフレームワーク関連ライブラリ
#-------------------------------------------
#flask関連ライブラリ
from flask import Flask, request, abort
#flaskインスタンス生成
app = Flask(__name__)

#-------------------------------------------
#LINE関連ライブラリ
#-------------------------------------------
#LINE API
from linebot import (
    LineBotApi, WebhookHandler
)

#LINE署名不一致による例外イベントの定義
from linebot.exceptions import (
    InvalidSignatureError
)

#LINEイベント
from linebot.models import (
    MessageEvent, TextMessage, TextSendMessage,
    SourceUser, SourceGroup, SourceRoom,
    TemplateSendMessage, ConfirmTemplate, MessageTemplateAction,
    ButtonsTemplate, ImageCarouselTemplate, ImageCarouselColumn,
    URITemplateAction,
    PostbackTemplateAction, DatetimePickerTemplateAction,
    CarouselTemplate, CarouselColumn, PostbackEvent,
    StickerMessage, StickerSendMessage, LocationMessage, LocationSendMessage,
    ImageMessage, VideoMessage, AudioMessage, FileMessage,
    UnfollowEvent, FollowEvent, JoinEvent, LeaveEvent, BeaconEvent
)
```

```
59
60
61
62 #---------------------------------------------
63 #環境変数の読み込み
64 #---------------------------------------------
65 #LINEチャネルシークレットトークン環境変数読み込み
66 channel_secret = os.getenv('LINE_CHANNEL_SECRET', None)
67 #LINEチャネルアクセストークン環境変数読み込み
68 channel_access_token = os.getenv('LINE_CHANNEL_ACCESS_TOKEN', None)
69 #もしチャネルシークレットトークンが読み込みできなければプロセス停止
70 if channel_secret is None:
71     print('Specify LINE_CHANNEL_SECRET as environment variable.')
72     sys.exit(1)
73 #もしチャネルアクセストークンが読み込みできなければプロセス停止
74 if channel_access_token is None:
75     print('Specify LINE_CHANNEL_ACCESS_TOKEN as environment variable.')
76     sys.exit(1)
77
78 #---------------------------------------------
79 #LINE APIインスタンス生成
80 #---------------------------------------------
81 #LINEbotAPIインスタンス生成
82 line_bot_api = LineBotApi(channel_access_token)
83 #WebhookHandlerインスタンス生成
84 handler = WebhookHandler(channel_secret)
85
86 #---------------------------------------------
87 #ダウンロードファイルの保存先ディレクトリ
88 #---------------------------------------------
89 #ファイルパスを生成
90 static_tmp_path = os.path.join(os.path.dirname(__file__), 'static', 'tmp')
91
92 #ディレクトリ作成関数
93 def make_static_tmp_dir():
94     try:
95         #ディレクトリ作成
96         os.makedirs(static_tmp_path)
97
98     #例外処理
99     except OSError as exc:
100         #もしファイルもしくはディレクトリが存在していたら何もしない
101         if exc.errno == errno.EEXIST and os.path.isdir(static_tmp_path):
102             pass
103         #もし別要因によるものなら例外を通知する
104         else:
105             raise
106
107
108 #---------------------------------------------
109 #電子署名検証
110 #(LINEbotからの呼び出しがHTTPSのため必須)
111 #---------------------------------------------
112 @app.route("/callback",methods=['POST'])
113 def callback():
114     # HTTPリクエストのヘッダを取得
115     signature = request.headers['X-Line-Signature']
116
```

```
117     # HTTPリクエスト(POST)のボディを取得
118     body = request.get_data(as_text=True)
119     app.logger.info("Request body: " + body)
120 
121     # 署名を検証し、問題なければ@handler.addに定義している関数を呼び出す
122     try:
123         handler.handle(body, signature)
124     except InvalidSignatureError:
125         abort( 400 )
126 
127     return 'OK'
128 
129 #非常停止フラグ初期化
130 emgcy_flg = False
131 
132 #----------------------------------------------
133 #メッセージイベントメソッド
134 #----------------------------------------------
135 @handler.add(MessageEvent, message=TextMessage)
136 def handle_text_message(event):
137     #非常停止フラグ
138     global emgcy_flg
139 
140     #PLCCOMインスタンス生成
141     plc=PLCCOM("10.0.11.220", 5001)
142 
143     #LINEから受信した文字を代入
144     text = event.message.text
145 
146 
147 
148 
149     if text=="設備を稼働させます！":
150         print(f"emgcy_flg:{emgcy_flg}")
151 
152         res = plc.read_machine_state()
153         if res=="STOP":
154             emgcy_flg=False
155 
156         if emgcy_flg==False:
157             res = plc.drive_machine()
158             if res==True:
159                 #「設備を稼働します」をmessageにセット
160                 Message = TextSendMessage(text="設備を稼働します")
161 
162             else:
163                 #「設備はすでに稼働中です」をmessageにセット
164                 Message = TextSendMessage(text="設備はすでに稼働中です")
165 
166         else:
167             #「設備は非常停止中です」をmessageにセット
168             msg = "設備は非常停止中です¥r¥n" + ¥
169                     "エラー要因を取り除きリセットボタンを押してください"
170             Message = TextSendMessage(text=msg)
171 
172     elif text=="設備を停止させます！":
173         res = plc.stop_machine()
174         if res==True:
```

```
175             #「設備を停止します」をmessageにセット
176             Message = TextSendMessage(text="設備を停止します")
177 
178         else:
179             #「設備はすでに停止中です」をmessageにセット
180             Message = TextSendMessage(text="設備はすでに停止中です")
181 
182     elif text=="設備を非常停止させます！":
183         res = plc.emergency_machine()
184         if res==True:
185             #「設備を非常停止中します」をmessageにセット
186             Message = TextSendMessage(text="設備を非常停止します")
187             #非常停止フラグをTrue
188             emgcy_flg=True
189         else:
190             #「設備はすでに非常停止中です」をmessageにセット
191             Message = TextSendMessage(text="設備はすでに非常停止中です")
192 
193     elif text=="設備をリセットさせます！":
194         res = plc.reset_machine()
195         if res==True:
196             #「設備をリセットします」をmessageにセット
197             Message = TextSendMessage(text="設備をリセットします")
198             #非常停止フラグをTrue
199             emgcy_flg=False
200         else:
201             #「設備はすでに稼働中です」をmessageにセット
202             msg = "設備はすでに稼働中です¥r¥n" + ¥
203                     "非常停止が作動した後にリセットしてください"
204             Message = TextSendMessage(text="設備はすでに稼働中です")
205 
206     elif text=="稼働状況を確認します！":
207         #機械の稼働状況を取得
208         state = plc.read_machine_state()        #稼働状況
209         pdt = plc.read_product()                #生産数
210         gd_pdt = plc.read_good_product()        #良品数
211         er_pdt = plc.read_defective_product()   #不良品の数
212 
213         #機械の稼働状況を判断、メッセージをセット
214         print(f"state:{state}")
215         if state=="STOP":
216             m_state = "現在停止中です¥r¥n"
217         elif state=="DRIVE":
218             m_state = "現在稼働中です¥r¥n"
219         elif state=="EMERGENCY":
220             m_state = "現在非常停止中です¥r¥n"
221         else:
222             m_state = "システムエラーが発生しました"
223 
224         msg = m_state + ¥
225                 "---------------¥r¥n" + ¥
226                 f"生産数:{pdt}¥r¥n" + ¥
227                 f"良品数:{gd_pdt}¥r¥n" + ¥
228                 f"不良品:{er_pdt}"
229 
230         Message = TextSendMessage(text=msg)
231 
232     elif text=="コマンド":
```

```
233         Message = TemplateSendMessage(
234             alt_text="Facility",
235             template=ButtonsTemplate(
236                 text="操作コマンドを選択してください",
237                 title="搬送負荷装置の制御",
238                 actions=[
239                     MessageTemplateAction(
240                         label="稼働",
241                         text="設備を稼働させます！"
242                     ),
243 
244                     MessageTemplateAction(
245                         label="停止",
246                         text="設備を停止させます！"
247                     ),
248 
249                     MessageTemplateAction(
250                         label="非常停止",
251                         text="設備を非常停止させます！"
252                     ),
253 
254 
255 
256                     MessageTemplateAction(
257                         label="稼働状況確認",
258                         text="稼働状況を確認します！"
259                     )
260 
261 
262                     #MessageTemplateAction(
263                     #   label="リセット",
264                     #   text="設備をリセットさせます！"
265                     #),
266 
267 
268                 ]
269             )
270         )
271 
272     else:
273         #「わかりません」をmessageにセット
274         Message=TextSendMessage("わかりません")
275 
276     #messageをLINEで送り返す
277     line_bot_api.reply_message(
278         event.reply_token,          #リクエスト応答用トークン
279         Message             #メッセージ送信
280     )
281 
282 if __name__== "__main__":
283     #ダウンロード先ディレクトリの生成
284     make_static_tmp_dir()
285 
286     #Flask起動
287     app.run(debug=True, port=8080)
288 
```

```
import smbus
"""
----------------------------------------------------------------------
温度センサ(TMP102)モジュール  v1.1

内　　容:オンボード温度センサ(TMP102)のライブラリモジュール
ファイル名:tmp102.py

@イニシャライザ
  TMP102() -- オブジェクト生成

@メソッド一覧
  readTemperature -- センサから温度情報の取得

----------------------------------------------------------------------
"""

TEMPERATURE_REG = 0x00
CONFIG_REG = 0x01
T_LOW_REG = 0x02
T_HIGH_REG = 0x03

ADDRESSES = [0x48, 0x49, 0x4A, 0x4B]

tempConvert = {
    'C': lambda x: x,
    'K': lambda x: x+273.15,
    'F': lambda x: x*9/5+32,
    'R': lambda x: (x+273.15)*9/5
}

tempConvertInv = {
    'C': lambda x: x,
    'K': lambda x: x-273.15,
    'F': lambda x: (x-32)*5/9,
    'R': lambda x: (x*5/9)-273.15
}
class TMP102(object):
    def __init__(self, units=None, address=0x48, busnum=1):
        units = units or 'C'
        if (address not in ADDRESSES):
            raise ValueError("Invalid Address: {0:#x}".format(address))
        self.address = address
        self.busnum = busnum

        self.setUnits(units)
        self.bus = smbus.SMBus(self.busnum)
        self.readTemperature()

    def bytesToTemp(self, data):
        # Adjustment for extended mode
        ext = self.extractConfig(1, 4, 1)
        #ext = data[1] & 0x01
        res = int((data[0] << (4+ext)) + (data[1] >> (4-ext)))

        if (data[0] | 0x7F is 0xFF):
            # Perform 2's complement operation (x = x-2^bits)
            res = res - 4096*(2**ext)
```

```
        # Outputs temperature in degC
        return res*0.0625

    def tempToBytes(self, temp):
        # Temp MUST be converted prior to input
        data = [0 , 0]
        res = int(temp/0.0625)
        ext = self.extractConfig(1, 4, 1)
        if (res < 0):
            res = res + 4096 * (2**ext)
        data[0] = (res >> (4 + ext)) & 0xFF
        data[1] = ((res & (2**(4 + ext)-1)) << (4 - ext)) | ext
        return data

    def extractConfig(self, num, location=0, length=0):

        data = self.bus.read_i2c_block_data(self.address, CONFIG_REG, 2)
        if (num == 3):
            #Full register dump
            return data
        else:
            mask = 2**length - 1
            return (data[num] >> location) & mask

    def injectConfig(self, setting, num, location, length):
        mask = (2**length - 1) << location
        setting = (setting << location) & mask
        data = self.bus.read_i2c_block_data(self.address, CONFIG_REG, 2)
        data[num] &= ~mask
        data[num] |= setting
        self.bus.write_i2c_block_data(self.address, CONFIG_REG, data)

    def readTemperature(self, units=None):
        data = self.bus.read_i2c_block_data(self.address, TEMPERATURE_REG, 2)
        tempC = self.bytesToTemp(data)
        units = units or self.units

        try:
            tempOut = tempConvert[units](tempC)
        except:
            raise ValueError('Invalid Units "' + self.units + '"')
        return tempOut

    def getUnits(self):
        return self.units

    def setUnits(self, units):
        if (units.upper() in 'RCKF' and len(units) == 1):
            self.units = units.upper()
        else:
            raise ValueError("Invalid Unit, must use C(elcius), K(elvin),"
                    "F(ahrenheit), or R(ankine)")



    def setConversionRate(self, rate):
        # 0 : 0.25 Hz
        # 1 : 1 Hz
```

```
        # 2 : 4 Hz (default)
        # 3 : 8 Hz
        self.injectConfig(rate, 1, 6, 2)

    def setExtendedMode(self, mode):
        # 0 : 12-bit ( -55C to 128C)
        # 1 : 13-bit ( -55C to 150C)

        self.injectConfig(mode, 1, 4, 1)

    def sleep(self):
        self.injectConfig(True, 0, 0, 1)

    def wakeup(self):
        self.injectConfig(False, 0, 0, 1)

    def setAlertPolarity(self, polarity):
        # 0 : Active Low
        # 1 : Active High
        self.injectConfig(polarity, 0, 2, 1)

    def alert(self):
        return extractConfig(1, 5, 1)

    def setFault(self, faultSetting):
        # 0 : 1 fault
        # 1 : 2 faults
        # 2 : 4 faults
        # 3 : 6 faults
        self.injectConfig(faultSetting, 0, 3, 2)

    def setAlertMode(self, mode):
        # 0 : Comparator Mode (Active within temp range)
        # 1 : Thermostat Mode (Active if over T_High, reset on read)
        self.injectConfig(mode, 0, 1, 1)

    def setBoundTemp(self, upper, temperature, units=None):
        units = units or self.units
        ext = self.extractConfig(1, 4, 1)
        try:
            temperature = tempConvertInv[units](temperature)
        except:
            raise ValueError('Invalid Units "' + self.units + '"')
        if (ext is 1 and temperature > 150):
            temperature = 150
        elif (temperature < -55):
            temperature = -55
        data = self.tempToBytes(temperature)

        if (upper):
            reg = T_HIGH_REG
        else:
            reg = T_LOW_REG

        self.bus.write_i2c_block_data(self.address, reg, data)

    def getBoundTemp(self, upper, units=None):
        units = units or self.units
```

```
175         if (upper):
176             reg = T_HIGH_REG
177         else:
178             reg = T_LOW_REG
179         data = self.bus.read_i2c_block_data(self.address, reg, 2)
180         tempC = self.bytesToTemp(data)
181 
182         try:
183             tempOut = tempConvert[units](tempC)
184         except:
185             raise ValueError('Invalid Units "' + self.units + '"')
186         return tempOut
```

```
import socket
import time

"""
----------------------------------------------------------------
PLCソケット通信モジュール　v1.1

内　　容:ＦＡ実習装置にあるＰＬＣの遠隔監視・制御用モジュール
ファイル名:PLCCOM.py
ＰＬＣ型番:FX3U-32M
　メーカー:三菱

メソッド一覧：
 @イニシャライザ
  PLCCOM(IPAddr ,port) -- オブジェクト生成

 @各種設定取得
  read_machine_state -- ＦＡ実習装置の状態取得
  read_product_target  -- 生産目標数取得
  read_product  -- 生産数取得
  read_good_product  --  生産数（良品）取得
  read_defective_product  --  生産数（不良品）取得

 @各種設定
  write_product_target  --  生産目標書き込み
  clear_all_product  --  生産数クリア
  clear_good_product -- 生産数（良品）クリア
  clear_defective_product -- 生産数（不良品）クリア

 @各種制御
  drive_machine -- 生産設備起動
  stop_machine -- 生産設備停止
  emergency_machine -- 生産設備緊急停止
  reset_machine -- 生産設備リセット

----------------------------------------------------------------
"""
class PLCCOM():
    PC="FF"             #PC番号
    TIMER = "0001"      #監視タイマ（書き込み／読み込み終了までの待機時間:250msec)
    endcode = "00"      #終了コード

    def __init__(self, IPAddr, port):
        server_ip = [IPAddr, port]
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client.connect(tuple(server_ip))
        print("PLCと接続しました")

    def __del__(self):
        self.client.close()
        print("PLCと切断しました")

    # デバイスコード生成
    def __create_device_code(self,dev,Bit_Word="BIT",option="count_value"):
        try:
            self.dev = dev.upper()
        except:
            print("__create_device:デバイスエラー：[X(x)][Y(y)][M(m)][T(t)][C(c)][D(d)]")
```

```
59              return False
60
61          try:
62              self.bit_word=Bit_Word.upper()
63          except:
64              print("__create_device:種別エラー：[BIT(bit)][WORD(word)]")
65              return False
66
67          if self.dev=="X":
68              return "5820"
69          elif self.dev=="Y":
70              return "5920"
71          elif self.dev=="M":
72              return "4D20"
73          elif self.dev=="T":
74              try:
75                  self.opt=option.upper()
76              except:
77                  print("__create_device:オプションエラー")
78                  print("option:[count_value] [point] [inductor]")
79                  return False
80              else:
81                  if option=="COUNT_VALUE":
82                      return "544E"
83                  elif option=="POINT":
84                      return "5453"
85                  elif option=="INDUCTOR":
86                      return "5443"
87          elif self.dev=="C":
88              try:
89                  self.opt=option.upper()
90              except:
91                  print("__create_device:オプションエラー")
92                  print("option:[count_value] [point] [inductor]")
93                  return False
94              else:
95                  if option=="COUNT_VALUE":
96                      return "434E"
97                  elif option=="POINT":
98                      return "4353"
99                  elif option=="INDUCTOR":
100                     return "4420"
101         elif self.dev=="D":
102             return "4420"
103
104
105     #送信パケット生成
106     def __create_RequestCode(self):
107         #リクエストコード生成
108         self.RequestCode = self.headcode + ¥
109                            PLCCOM.PC + ¥
110                            PLCCOM.TIMER + ¥
111                            self.DeviceCode + ¥
112                            f"{self.DeviceNumber:08x}" + ¥
113                            f"{self.Count:02x}"+ ¥
114                            PLCCOM.endcode
115
116         return self.RequestCode
```

```
117
118     #-----------------------------------------------
119     #read_device  --  データ読み取り
120     #param:
121     #  SubHeader -- サブヘッダ("00":ビット読み出し,"01":ワード読み出し)
122     #  Device -- デバイス(X,Y,M,T,C,D)と番号3桁(000～999)
123     #  Count -- デバイス数
124     #return
125     #  False -- 引数エラー,  false以外  --  応答コード
126     #-----------------------------------------------
127     def read_device(self, SubHeader,Device,Count):
128
129
130         self.headcode=SubHeader #サブヘッダをセット
131         self.DeviceCode = self.__create_device_code(Device[0]) #デバイスコードセット
132         if self.DeviceCode==False:
133             return False
134         try:
135             self.DeviceNumber = int(Device[1:4])    #デバイス番号セット
136         except:
137             return False
138         else:
139             self.Count = Count  #デバイス数セット
140
141             #リクエストコード生成
142             self.RequestCode = self.__create_RequestCode()
143
144             #リクエスト送信
145             self.client.sendall(self.RequestCode.encode())
146
147             #レスポンス受信
148             self.response=self.client.recv(512).decode()
149
150             return self.response
151
152     #-----------------------------------------------
153     #write_device_bit  --  ビットデータ書き込み
154     #param:
155     #  Device -- デバイス(X,Y,M,T,C,D)と番号3桁(000～999)
156     #  Count -- デバイス数
157     #  data -- ビットデータ (リスト型)
158     #return:
159     #  False -- 引数エラー, False以外 -- 応答コード
160     #-----------------------------------------------
161     def write_device_bit(self, Device,Count,data):
162         self.headcode="02"
163         self.DeviceCode = self.__create_device_code(Device[0]) #デバイスコードセット
164         if self.DeviceCode==False:
165             return False
166         try:
167             self.DeviceNumber = int(Device[1:4])    #デバイス番号セット
168         except:
169             return False
170         else:
171             self.Count = Count  #デバイス数セット
172
173
174             #リクエストコード生成
```

```
175             self.RequestCode = self.__create_RequestCode()
176             for i in range(Count):
177                 self.RequestCode+=f"{data[i]}"
178 
179             #リクエスト送信
180             self.client.sendall(self.RequestCode.encode())
181 
182             #レスポンス受信
183             self.response=self.client.recv(512).decode()
184 
185             return self.response
186 
187     #-------------------------------------------------
188     #write_device_word  --  ワードデータ書き込み
189     #param:
190     #  Device -- デバイス(X,Y,M,T,C,D)と番号3桁(000～999)
191     #  Count -- デバイス数
192     #  data -- ワードデータ (リスト型)
193     #return:
194     #  応答コード
195     #-------------------------------------------------
196     def write_device_word(self, Device,Count,data):
197         self.headcode="03"
198         self.DeviceCode = self.__create_device_code(Device[0]) #デバイスコードセット
199         if self.DeviceCode==False:
200             return False
201         try:
202             self.DeviceNumber = int(Device[1:4])    #デバイス番号セット
203         except:
204             return False
205         else:
206             self.Count = Count  #デバイス数セット
207 
208 
209             #リクエストコード生成
210             self.RequestCode = self.__create_RequestCode()
211             for i in range(Count):
212                 self.RequestCode+=f"{data[i]:04x}"
213 
214             #リクエスト送信
215             self.client.sendall(self.RequestCode.encode())
216 
217             #レスポンス受信
218             self.response=self.client.recv(512).decode()
219 
220             return self.response
221 
222 
223     #-------------------------------------------------
224     #read_machine_state  -- 機械の状態を読み込み
225     #param:
226     #   None
227     #return:
228     #   "STOP"    -- 停止中
229     #   "DRIVE"   -- 運転中
230     #   "EMERGENCY" -- 緊急停止中
231     #   "RESET"   -- 原点復帰中
232     #-------------------------------------------------
```

```
233     def read_machine_state(self):
234         self.res = self.read_device("00","M200",4)
235         print(f"self.res:{self.res}")
236         if self.res[4:8]=="1000":
237             return "STOP"
238         elif self.res[4:8]=="0100":
239             return "DRIVE"
240         elif self.res[4:8]=="1010":
241             return "EMERGENCY"
242         elif self.res[4:8]=="1011":
243             return "RESET"
244     #-------------------------------------------------
245     #read_product_target -- 生産目標読み込み
246     #param:
247     #   None
248     #return:
249     #   生産目標数
250     #-------------------------------------------------
251     def read_product_target(self):
252         self.res = self.read_device("01","D200",1)
253         return int(self.res[4:8],16) #PLCからのレスポンスが16進数なので10進数へ変換する
254 
255     #-------------------------------------------------
256     #read_product -- 生産数読み込み
257     #param:
258     #   None
259     #return:
260     #   生産数（良品＋不良品）
261     #-------------------------------------------------
262     def read_product(self):
263         self.res = self.read_device("01","D201",1)
264         return int(self.res[4:8],16) #PLCからのレスポンスが16進数なので10進数へ変換する
265 
266     #-------------------------------------------------
267     #read_good_product -- 生産数（良品数）読み込み
268     #param:
269     #   None
270     #return:
271     #   生産数（良品）
272     #-------------------------------------------------
273     def read_good_product(self):
274         self.res = self.read_device("01","D202",1)
275         return int(self.res[4:8],16) #PLCからのレスポンスが16進数なので10進数へ変換する
276 
277     #-------------------------------------------------
278     #read_good_product -- 生産数（不良品数）読み込み
279     #param:
280     #   None
281     #return
282     #   生産数（不良品）
283     #-------------------------------------------------
284     def read_defective_product(self):
285         self.res = self.read_device("01","D203",1)
286         return int(self.res[4:8],16) #PLCからのレスポンスが16進数なので10進数へ変換する
287 
288     #-------------------------------------------------
289     #write_product_target -- 生産目標書き込み
290     #param:
```

```
291     #   value -- 生産目標数 (0以上)
292     #return:
293     #   True -- 正常終了,  False  --  引数エラー
294     #--------------------------------------------------
295     def write_product_target(self, value):
296         if value>=0:
297             self.res = self.write_device_word("D300", 1,[value])
298             return True
299         else:
300             return False
301 
302     #--------------------------------------------------
303     #clear_all_product -- 生産数クリア
304     #param:
305     #   なし
306     #return:
307     #   なし
308     #--------------------------------------------------
309     def clear_all_product(self):
310         self.res = self.clear_good_product()
311         self.res = self.clear_defective_product()
312 
313     #--------------------------------------------------
314     #clear_good_product -- 生産数 (良品) クリア
315     #param:
316     #   なし
317     #return:
318     #   なし
319     #--------------------------------------------------
320     def clear_good_product(self):
321         self.res = self.write_device_word("D302",1,[0])
322 
323     #--------------------------------------------------
324     #clear_defective_product -- 生産数 (不良品) クリア
325     #param:
326     #   なし
327     #return:
328     #   なし
329     #--------------------------------------------------
330     def clear_defective_product(self):
331         self.res = self.write_device_word("D303",1,[0])
332 
333     #--------------------------------------------------
334     #drive_machine -- 生産設備起動
335     #param:
336     #   なし
337     #return:
338     #   True -- 正常終了, False -- すでに運転中
339     #--------------------------------------------------
340     def drive_machine(self):
341         if self.read_machine_state() == "DRIVE":
342             return False
343         else:
344             self.res = self.write_device_bit("x000", 4, [1, 0, 0, 0])
345             return True
346 
347     #--------------------------------------------------
348     #stop_machine -- 生産設備停止
```

```
349     #param:
350     #   なし
351     #return:
352     #   True -- 正常終了, False -- すでに停止中
353     #Note:
354     #  すでにロボットアームによる搬送が行われている場合は仕分けが完了するまで
355     #  動作を継続する
356     #-------------------------------------------------
357     def stop_machine(self):
358         if self.read_machine_state() == "STOP":
359             return False
360         else:
361             self.res = self.write_device_bit("x000", 4, [0, 1, 0, 0])
362             return True
363 
364     #-------------------------------------------------
365     #emergency_machine -- 生産設備非常停止
366     #param:
367     #   なし
368     #return:
369     #   True -- 正常終了, False -- すでに停止中
370     #-------------------------------------------------
371     def emergency_machine(self):
372         if self.read_machine_state() == "EMERGENCY":
373             return False
374         else:
375             self.res = self.write_device_bit("x000", 4, [0, 0, 0, 1])
376             return True
377 
378     #-------------------------------------------------
379     #reset_machine -- 生産設備リセット
380     #param:
381     #   なし
382     #return:
383     #   True -- 正常終了, False -- 運転中のためリセット不可能
384     #Note:
385     #  すでに生産設備が運転中の場合はリセット不可能。停止中もしくは非常停止中のとき
386     #  生産数をクリアし、ロボットアームを原点に復帰させる。
387     #-------------------------------------------------
388     def reset_machine(self):
389         if self.read_machine_state() == "DRIVE":
390             return False
391         else:
392             self.res = self.write_device_bit("x000", 4, [0, 0, 1, 0])
393             return True
394 
395 if __name__=="__main__":
396     plc = PLCCOM("10.0.11.220", 5001)
397 
398 
399     print("-------------------------")
400     print("現在の設定値")
401     print("-------------------------")
402     res = plc.read_machine_state()
403     print("state:"+res)
404     res = plc.read_product_target()
405     print(f"生産目標:{res}")
406     res = plc.read_product()
```

```
    print(f"生産数:{res}")
    res = plc.read_good_product()
    print(f"良品数:{res}")
    res = plc.read_defective_product()
    print(f"不良品数:{res}")

    """
    print("-------------------------")
    print("生産目標数をセットしました")
    print("-------------------------")
    res = plc.write_product_target(80)
    plc.clear_all_product()
    res = plc.read_product_target()
    print(f"生産目標:{res}")
    res = plc.read_product()
    print(f"生産数:{res}")
    res = plc.read_good_product()
    print(f"良品数:{res}")
    res = plc.read_defective_product()
    print(f"不良品数:{res}")
    """
    plc.stop_machine()

    print("-------------------------")
    print("設備を動作します")
    print("-------------------------")
    plc.drive_machine()

    res = plc.read_machine_state()
    print("state:"+res)
    time.sleep(8.0)


    print("-------------------------")
    print("設備を緊急停止します")
    print("-------------------------")
    plc.emergency_machine()
    time.sleep(30.0)

    res = plc.read_machine_state()
    print("state:"+res)

    print("-------------------------")
    print("設備をリセットします")
    print("-------------------------")
    plc.reset_machine()


    res = plc.read_machine_state()
    print("state:"+res)
    time.sleep(10.0)
    res = plc.read_machine_state()
    print("state:"+res)
```