第３章　ＳＮＳを利用したＩｏＴアプリケーション開発

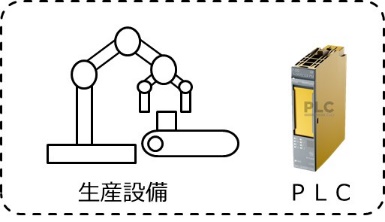
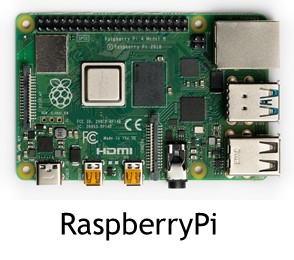
**３．１****製造現場におけるＳＮＳ活用**

　ＳＮＳ（Social Networking Service）には様々なものがあり，特定の人物との会話や不特定多数の相手との連絡手段，飲食店や商業施設のＰＲなど様々な活用がされています．

　ＳＮＳの中にはＡＰＩ(Application Programable Interface)が公開されているものもあり，他のアプリケーションと連携させることが容易となっているものもあります．このようにＡＰＩを活用することで生産現場におけるＳＮＳ活用の幅が広がる可能性があります．

例として製造装置の遠隔制御・監視方法を考えてみます．遠隔地から特定の制御装置には制御用コンピュータ（ＰＬＣ）にアクセスする必要があります．制御用コンピュータはネットワークからソケットを使用した特定のコマンドを受信することで内部レジスタや接点情報を取得することができます． この場合，コマンド送受信をするためのアプリケーションがインストールされているコンピュータしかアクセスできないためセキュリティ面では強固である反面，インターフェース設計など他のプログラム言語などの知識を要します．

　一方，遠隔制御・監視手段としてＳＮＳを活用すると操作方法などは普段の使用方法と変わらないため時間はかからないですし，外部ネットワークに接続するためローカルエリア内の端末以外の端末からアクセスすることも可能です．またスマートフォンの通知機能と併用すると緊急事態における端末へのサウンド、バイブレーションを活用した即時を行うことができるため迅速な対応もしやすくなります．



**通知**

**センシング**

**メッセージ**



**制御信号**



**SNS**

図３．１　ＳＮＳ活用のイメージ

本実習で使用するSNSはLINE🄬です．LINEは国内で使用されているSNSとして広く知られているだけでなく，LINE botという仕組みが公開されておりAPIを経由して誰でも簡単に自動コミュニケーションツールを活用することができます．LINE botを使用することでメッセージやデータ通知を利用して自動的に行うことができます．



LINE



Webhook

ngrok

LINE Platform

User

Flask

Device

Webhook

handler

アイコン

自動的に生成された説明

Application

Senser

図３．２　LINE botの仕組み

　ユーザーがLINEでメッセージを送るとwebhookに登録されたアドレスにHTTPSリクエストを送ります．外部公開されているURLをローカルIPに変換し(ngrok)，webフレームワーク(Flask)のコールバックルーチンを呼び出します．コールバックルーチンではLINEが提供しているMessagingAPIを利用して送信されたメッセージをアプリケーション上で使用することができます．メッセージ内容によってデバイス制御やセンサ取得ができます。

Webhook: アプリケーションの更新情報を他のアプリケーションへリアルタイム提供する仕組み

ngrok:通常はローカル環境でアクセスできるURLを外部公開するサービス

flask:Pythonで扱うことができる軽量Webフレームワーク

Webhookhandler:Webhookからのイベントを受信したときに実行する処理(コールバック)

**３．２　ＬＩＮＥ　ＡＰＩの環境構築**

(1)LINE アカウント作成

①　　　　　　　　　②　　　　　　　　③　　　　　　　　 ④



⑤　　　　　　　　 ⑥　　　　　　　　 ⑦　　　　　　　　 ⑧



⑨　　　　　　　　　⑩　　　　　　　　⑪　　　　　　　　⑫

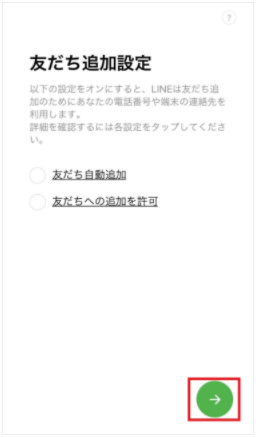
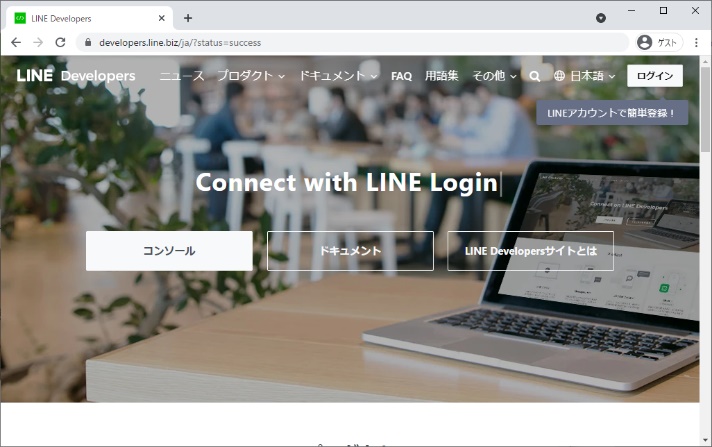


図３．３　ＬＩＮＥアカウントの取得

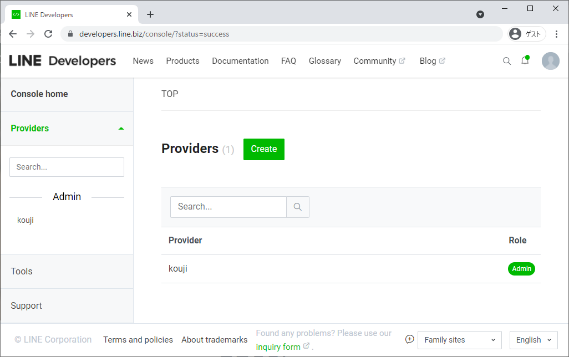
(2)LINE Developerのログイン

URL:<https://developers.line.biz/ja/>

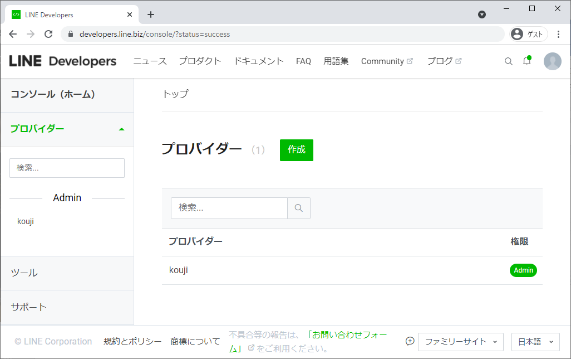
①　　　　　　　　　　　　　　　　　　　　　　　　②



③　　　　　　　　　　　　④　　　　　　　　　　　⑤



⑥　　　　　　　　　　　　　　　　　　　　　　　　⑦



日本語表記に変更



図３．４　LINE Developerへのログイン

(3)プロバイダー作成

LINEプラットフォームを通じてサービスを提供する個人や企業を登録します．

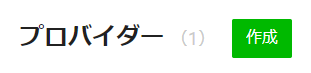
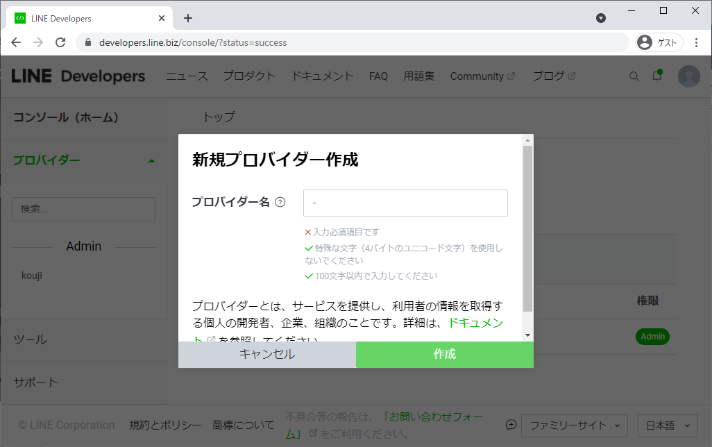


図３．５　新規プロバイダー作成

(4)MessagingAPIチャネル作成

MessagingAPIとは，LINEプラットフォームからのメッセージの通知もしくはプログラムコードで生成したメッセージをLINEプラットロームに送信するAPIのことです．Messaging APIで提供されるサービスには下記のものがあります．

1. 応答メッセージを送る
2. プッシュメッセージを送る
3. その他さまざまなタイプのメッセージ（テキスト、スタンプ、画像、動画、音声、位置情報、イメージマップ、テンプレート）
4. ユーザーが送ったコンテンツを取得する
5. ユーザープロフィールを取得する
6. グループチャットに参加する
7. リッチメニューを使う
8. ビーコンを使う
9. アカウント連携を使う
10. 送信メッセージ数を取得する

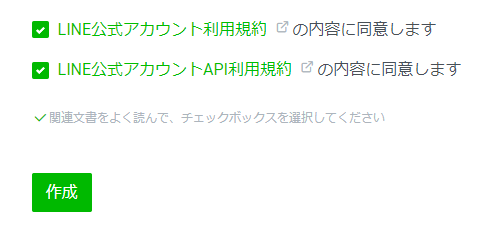
MessagingAPIを使用するには下記の順にMessagingAPIチャネルを作成します．

①



チャネル名（必須）大業種（必須）、小業種（必須）、利用規約への同意を記入する．

②　　　　　　　　　　　　③　　　　　　　　　　　　　　　　④



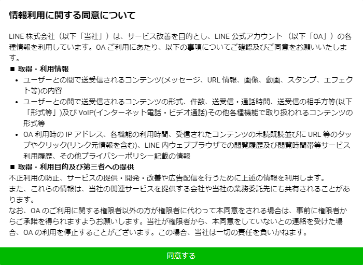




図３．６　MessagingAPIチャネル作成

(5)環境変数にチャネルアクセストークン，シークレットトークンを設定する．

　LINE Developerのシークレットトークンとチャネルアクセストークンを環境変数へ代入します．各トークンは作成したチャンネルの基本設定およびMessaging API設定の画面で確認できます．

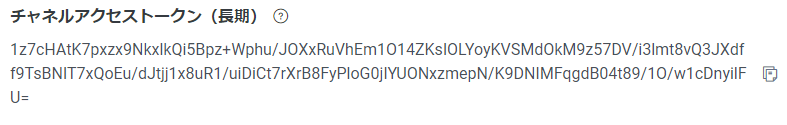




※チャネルによってトークンが異なる

図３．７　シークレットトークンの確認





※チャネルによってトークンが異なる

図３．８　チャネルアクセストークンの確認

①環境変数へ代入

|  |
| --- |
| $ nano ~/.bashrc |

②.bashrcの末尾に追加する

図３．７と図３．８で確認した各トークンをコピー＆ペーストします．

|  |
| --- |
| #シークレットトークン  export LINE\_CHANNEL\_SECRET=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #チャネルアクセストークン  export LINE\_CHANNEL\_ACCESS\_TOKEN=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

③もし社内プロキシを採用しているネットワークの場合は同一の.bashrcに下記の記述を追加します．

|  |
| --- |
| export HTTP\_PROXY=http://<user>:<password>@<proxyserver>:<port>  export HTTPS\_PROXY=https://<user>:<password>@<proxyserver>:<port> |

ポリテクセンター山梨ではプロキシサーバを採用しているため下記の記述のように末尾に記載します．

|  |
| --- |
| export HTTP\_PROXY=http://10.0.0.2:15080  export HTTPS\_PROXY=https://10.0.0.2:15080 |

(6)LINE APIのインストール

|  |
| --- |
| $sudo pip3 --proxy=http://10.0.0.2:15080 install line-bot-sdk |

(7)ngrokダウンロードとインストール

　ngrok(エングロック)とは，ローカル環境で実行しているWEBアプリケーション(http,thhps,TCP)を外部公開するサービスです．ngrokには有償版と無償版があります．

有償版：プランによってひと月当たりの料金が変わります．公開されるURLを自由に設定できる（つまり固定化できる）などメリットがあります．

無償版：料金がかからない代わりに，公開されるURLを固定化することができません．ngrokを起動させるたびにURLがランダムに変わります．また，ダウンロードの際にユーザー登録しないと公開URLへの接続に時間制限が設けられます（２時間程度）

本実習ではngrok無償版を使用します．必要に応じて有償版にアップグレードできます．

1. 下記のURLからダウンロード・展開する

|  |
| --- |
| $wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-arm.zip  $unzip ngrok-stable-linux-arm.zip |

②ダウンロードした展開してラズベリーパイへコピー

コピー先： /usr/local/bin/

|  |
| --- |
| $sudo mv ./ngrok /usr/local/bin/ |

(9)ngrok実行（本実習ではポート番号を8080とします）

|  |
| --- |
| $ngrok http 8080 |

ngrokによって外部公開されるURL(https)を確認します．下記のように外部公開されたURLをLINE DeveloperのWebhook URLに登録します．

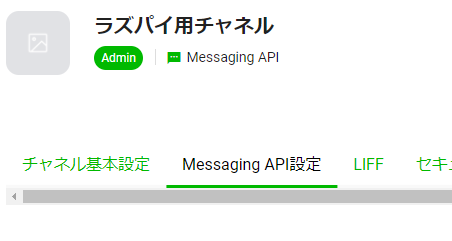
　TeraTarmを使用している場合，下記の網掛けになっているURLをドラックすることでクリップボードにコピーできます．

|  |
| --- |
| ngrok by @inconshreveable (Ctrl+C to quit)  Session Status online  Session Expires 1 hour, 57 minutes  外部公開されるURL(https)をコピーする！  Version 2.3.40  Region United States (us)  Web Interface http://127.0.0.1:4040  Forwarding http:// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.ngrok.io -> http://localhost:8080  Forwarding https://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.ngrok.io -> http://localhost:8080  Connections ttl opn rt1 rt5 p50 p90  0 0 0.00 0.00 0.00 0.00 |

※このURLはngrok（無料版）が起動するたびにランダムで与えられます．

図３．９　ngrok起動画面

(10)Webhook URLに登録する



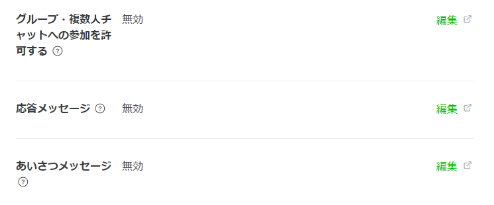
ngrokで公開されたURL(https)を貼り付けし，末尾に「/callback」を追加する



図３．１０　WebhookURLの設定

(11)応答メッセージを設定する

　初期状態ではLINEbotが受信したメッセージに対して自動返信する設定となっているため無効にします．



「応答メッセージ」－「編集」

「あいさつメッセージ」をオフ

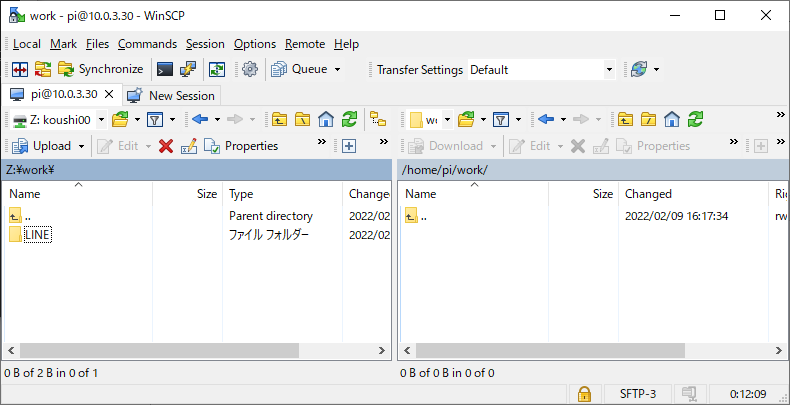
「応答メッセージ」をオフ

図３．１１　応答メッセージ設定

(12)メッセージイベントアプリをコピー

コピー元 z:\work\LINE

コピー先 /home/pi/work/

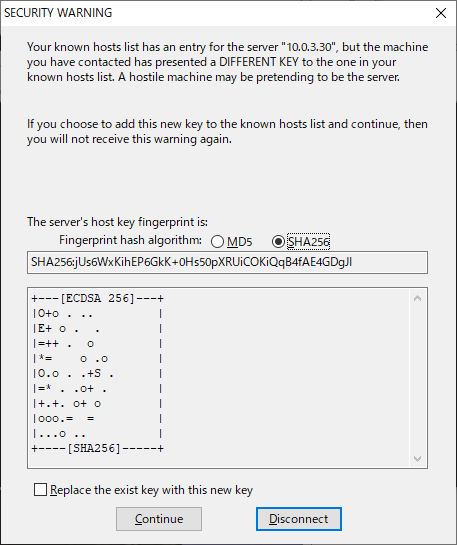


コピー＆ペースト

図３．１２　メッセージイベントアプリのコピー

(13)ターミナル画面を複製してメッセージイベントアプリを起動させる

①

[File]-[Duplicate session]をクリック

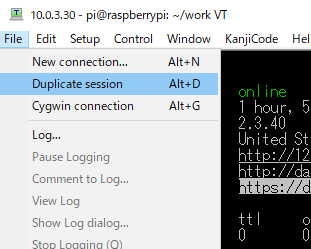


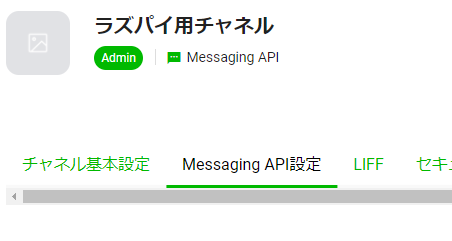
図３．１３　ターミナルの複製

②

|  |
| --- |
| $ cd ~/work/LINE  $ python3 ./messageEvent.py |

(14)QRコード読み取り

スマートフォンもしくはタブレットPCのLINEbotのQRコードを読み取ります



画面に表示されているQRコードを読み取り





図３．１４　チャネル登録

(15)作成したチャネルにメッセージを送る

メッセージを送ると同じメッセージが送り返されることを確認します。

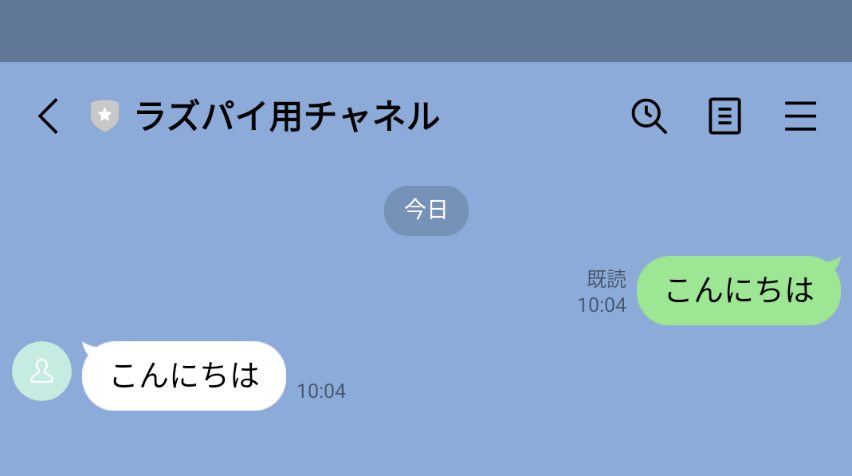


図３．１５　メッセージ送信

**３．３　ＬＩＮＥ　ＡＰＩ（メッセージイベント）**

サンプルコードを例にそれぞれのLINE APIを解説します．

(1)LINE bot ライブラリのインポート

LINE APIを使用するには下記のようにLINE botライブラリをインポートします．非常に多くのライブラリやモジュール（クラス）が存在します．LINEbotライブラリの中の２つのライブラリ(LineBotApi, WebhookHandler)をインポートします．

|  |
| --- |
| from linebot import (  LineBotApi, WebhookHandler  ) |

(2)各種イベントモジュールのインポート

|  |
| --- |
| from linebot.models import (  MessageEvent, TextMessage, TextSendMessage,  ……  ) |

WebhookHanlderメソッドに必要なイベントモジュールは下記の通りです．

linebot.models:LINEbotの各種イベントを定義しているライブラリ

MessageEvent:メッセージを受信した際に通知されるイベント

TextMessage:メッセージオブジェクト

TextSendMessage:LINEプラットフォームにメッセージ送信する関数

(3)チャネルアクセストークンとチャネルシークレットトークンを読み込み

チャンネルアクセストークンはあらかじめ環境変数に代入されているためコードのなかで読み込みします．

|  |
| --- |
| import os  ……  #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) |

(4)LINE bot API のインスタンス生成

　LINE bot APIのインスタンスを生成するにはチャンネルアクセストークンを必要とします．

|  |
| --- |
| line\_bot\_api = LineBotApi(channel\_access\_token) |

(5)WebhookHandlerインスタンス生成

WebhookHandlerのインスタンスを生成するにはチャネルシークレットトークンを必要とします．

|  |
| --- |
| handler = WebhookHandler(channel\_secret) |

(6)電子署名の検証ルーチン（コールバックルーチン）

　リクエストがLINEプラットフォームから送られてきたものなのか検証するため，イベントの通知をする前に電子署名を検証します．検証するためには「X-Line-Signature」が含まれているHTTPSリクエストヘッダを取得します．

|  |
| --- |
| @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']  # HTTPリクエスト(POST)のボディを取得  body = request.get\_data(as\_text=True)  app.logger.info("Request body: " + body)  # 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  try:  handler.handle(body, signature)  WebhookHandlerメソッドの呼び出し(署名の検証も同時に行う)  except InvalidSignatureError:  abort( 400 )  return 'OK' |

(7)WebhookHandlerメソッドの追加

自作関数をWebhookHandlerメソッドに追加するには@handler.addを記述します．

handler.addには通知するイベント(MessageEvent)とメッセージオブジェクト(TextMessage)を引数として渡します．メッセージオブジェクトは自作関数の引数(event)に引き継がれます．

|  |
| --- |
| @handler.add(MessageEvent, message=TextMessage)  def handl\_text\_message(event):  自作関数(WebhookHandler)に登録  …… |

(8)LINEからのメッセージ取得

LINEからメッセージを取得するにはHTTPリクエストボディから取得する必要があります．自作関数の引数(event)にはHTTPSリクエストボディが代入されています．

|  |
| --- |
| @handler.add(MessageEvent, message=TextMessage)  def handl\_text\_message(event):  #メッセージを取得  text = event.message.text  …… |

LINEからのメッセージは文字列で取得できます．従って，これ以降の処理は通常の文字列処理として分岐や解析を行うことが可能です．

(9)LINEへのメッセージ送信

　LINEに対してメッセージを送信するにはLINEbotAPIにおけるreply\_messageメソッドを使用します．reply\_messageメソッドには応答用トークンとメッセージ関数としてTextSendMessage関数を使用します．

|  |
| --- |
| @handler.add(MessageEvent, message=TextMessage)  def handl\_text\_message(event):  ……  #LINEにメッセージ送信  line\_bot\_api.reply\_message(  event.reply\_token, #リクエスト応答用トークン  TextSendMessage(text) #送信メッセージ関数（メッセージをセット）  ) |

送信メッセージは固定の文字列だけでなく，センサ情報を文字列に変換することで送信することができます．

|  |
| --- |
| @handler.add(MessageEvent, message=TextMessage)  def handl\_text\_message(event):  #環境センサ(BME280)から温度,気圧,湿度データの取得(float型)  tmp,pre,hum = bme280.readData()  #LINEにメッセージ送信  line\_bot\_api.reply\_message(  event.reply\_token, #リクエスト応答用トークン  TextSendMessage(f”{tmp:.1f}”) #温度データ(少数第1位)を文字列に変換しセットする  ) |

(10)メッセージ送受信シーケンス

ユーザーがLINEにメッセージを送信しメッセージが送り返されるまでの流れを下記の図に示します．ユーザーがメッセージを送信するとLINEからWebhookを通じてngrokで公開されている端末にHTTPSリクエストを送ります．Webフレームワークによってエンドポイントに指定されているコールバック関数が実行されます．コールバック関数では電子署名の検証とWebhookHandlerの呼び出しを行い，”OK”を戻します．登録されたWebhookHandlerでは受信メッセージの解析や周辺回路の制御を行い，必要に応じてメッセージを返します(line\_bot\_api.reply\_messageメソッド)．

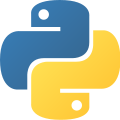


図３．１６　LINEbotにおけるシーケンス図

**３．４　練習問題（システム内部の温度取得）**

(1)　仕様

ラズベリーパイ用LINEチャネルに対して「温度を表示」とメッセージを表示すると「現在　〇〇℃です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示するLINEbotを作成しましょう．

(2)ファイル名とファイルパス

ファイル名：tempEvent.py

ファイルパス：/home/pi/work/LINE/

(3) 温度センサ

　本課題ではラズベリーパイに接続されているサンハヤト製拡張IOボードを使用します．実装されている温度センサTMP102から温度を取得し，LINEメッセージに送信しましょう．

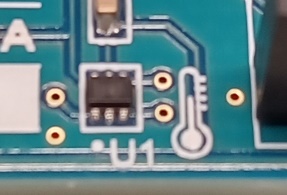


図３．１７　温度センサ(TMP102)

|  |
| --- |
| #温度センサライブラリのインポート  from tmp102 import TMP102  ……  #インスタンス生成  tmp = TMP102()  #温度取得(float型)  t=tmp.readTemperature() |

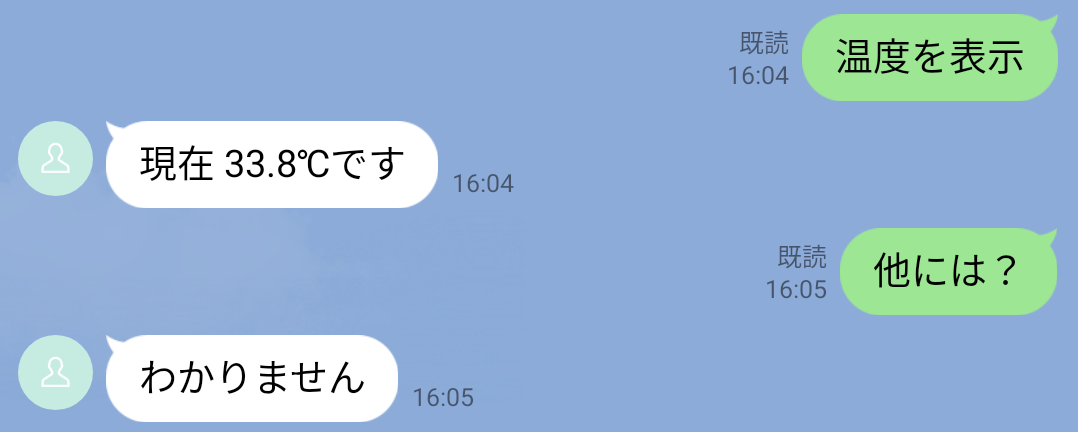
(4)実行例

図３．１８　システム温度の問い合わせ例

**３．５　搬送負荷装置ライブラリ**

　第２章で使用したMCプロトコルを使用し負荷装置の稼働状況を取得します．ただしMCプロトコルは要求伝文（パケット）が複雑であるため，LINEbotプログラムの中にすべて記述すると可読性が悪くなるうえ不具合発生時のデバッグに時間がかかります．

　本実習ではあらかじめ搬送負荷装置の各種情報の取得と設定をするための専用ライブラリを用意しました．

(1)ライブラリインポート

|  |
| --- |
| from PLCCOM import PLCCOM |

(2)ライブラリ解説

①インスタンス生成

|  |  |
| --- | --- |
| 関数名 | PLCCOM(IPAddress, PORT) |
| 引数 | IPAddress: PLCのIPアドレス(str型)  PORT: PLCのポート番号(int型) |
| 戻り値 | PLCと通信するインスタンス |
| 概要 | PLCと通信するインスタンスを生成する |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  … |

②稼働状況の取得

|  |  |
| --- | --- |
| 関数名 | read\_machine\_state() |
| 引数 | なし |
| 戻り値 | “STOP” : 停止中  “DRIVE” : 稼働中  “EMERGENCY” : 非常停止中  “RESET” : 原点復帰中 |
| 概要 | 搬送負荷装置の状態を取得する |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  #装置の状態を取得  state = plc.read\_machine\_state()  #もし機械が非常停止中だったら…  if state==”EMERGENCY”:  …… |

③生産目標数の取得

|  |  |
| --- | --- |
| 関数名 | read\_product\_target() |
| 引数 | なし |
| 戻り値 | 生産目標数(int型) |
| 概要 | 生産目標数を取得する |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  #装置の生産目標数を取得  ptget = plc.read\_product\_target()  #現在の生産目標数の表示  print(f“現在の目標数は{ptget}個です.”) |

④生産数の取得

|  |  |
| --- | --- |
| 関数名 | read\_product() |
| 引数 | なし |
| 戻り値 | 生産数(int型) |
| 概要 | 生産数を取得する．生産数とは良品の数と不良品の数を加算した値． |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  #装置の生産目標数を取得  pt = plc.read\_product()  #現在の生産目標数の表示  print(f“現在の生産数は{pt}個です.”) |

⑤良品数の取得

|  |  |
| --- | --- |
| 関数名 | read\_good\_product() |
| 引数 | なし |
| 戻り値 | 良品の数(int型) |
| 概要 | 良品の数を取得する． |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  #装置の良品数を取得  good\_pt = plc.read\_good\_product()  #現在の良品数の表示  print(f“現在の良品数は{good\_pt}個です.”) |

⑥不良品数の取得

|  |  |
| --- | --- |
| 関数名 | read\_defective\_product() |
| 引数 | なし |
| 戻り値 | 良品の数(int型) |
| 概要 | 良品の数を取得する． |
| 使用例 | from PLCCOM import PLCCOM  #インスタンス生成  plc = PLCCOM(“10.0.11.220”, 5001)  #装置の不良品を取得  defective\_pt = plc.read\_defective\_product()  #現在の不良品の表示  print(f“現在の良品数は{defective\_pt}個です.”) |

**３．６　練習問題（搬送負荷装置における稼働状況の問い合わせ）**

(1)　仕様

ラズベリーパイ用LINEチャネルに対して「稼働状況」とメッセージを送信すると搬送負荷装置の稼働状況を「現在〇〇です」と表示し，それ以外のメッセージを受信した場合「わかりません」と表示するLINEbotを作成しましょう．

(2)ファイル名とファイルパス

ファイル名：machineStateEvent.py

ファイルパス：/home/pi/work/LINE/

(3)実行例

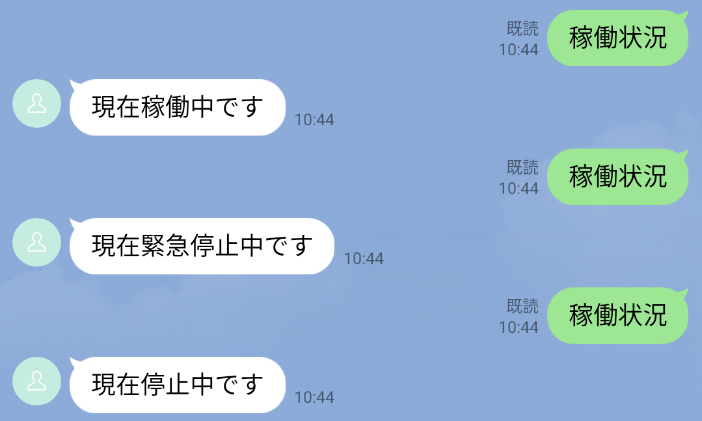


図３．１９　LINEによる稼働状況の問い合わせ例

(4)記述例

|  |
| --- |
| @handler.add(MessageEvent, message=TextMessage)  def handle\_text\_message(event):  #LINEから受信した文字を代入  text = event.message.text    if text=="稼働状況":  #搬送負荷装置ライブラリのインスタンス生成  plc=PLCCOM("???.???.???.???", ????)  #稼働状況の取得  state = plc.?????????????????()    #機械の稼働状況を判断、メッセージをセット  if state=="???????":　#もしstateが”STOP”だったら…  message = "現在停止中です"  elif state=="???????":　#もしstateが”DRIVE”だったら…  message = "現在稼働中です"  elif state=="???????":　#もしstateが”EMERGENCY”だったら…  message = "現在緊急停止中です"  else:  message = "システムエラー"    #稼働状況をLINEで送り返す  line\_bot\_api.reply\_message(  event.reply\_token, #リクエスト応答用トークン  TextSendMessage(text=message) #メッセージ送信  )    else:  #「わかりません」をLINEで送り返す  line\_bot\_api.reply\_message(  event.reply\_token, #リクエスト応答用トークン  TextSendMessage("わかりません") #メッセージ送信  ) |

(5)追加仕様

「稼働状況」のメッセージを送信すると搬送負荷装置の稼働状況を「現在〇〇です」と表示するとともに生産数，良品数，不良品数を表示するように改良しましょう．

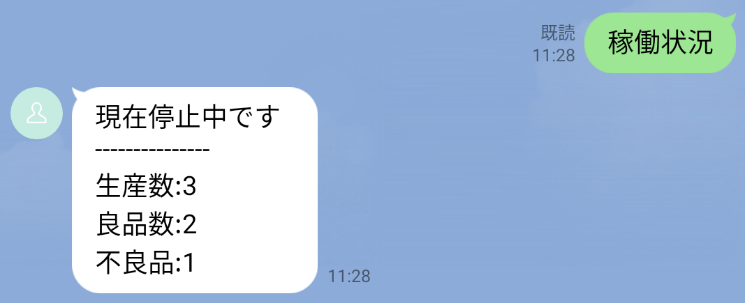


図３．２０　稼働状況と生産状況の問い合わせ例

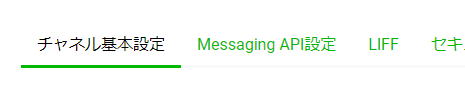
**３．７　通知機能の実装**

　これまでのLINEを使用した稼働状況の取得では，ユーザーが問い合わせをすることでLINEbotが応答する形式となっていました．ところがシステムエラーや何らかの要因により設備が緊急停止する状況が発生した場合は，問い合わせを待つことなく状況を通知することで手早くエラー要因を特定することができます．

　LINEbotAPIにはメッセージを通知するプッシュ通知機能が実装されています．メッセージイベントのプログラムコードと比較して少ないコード量で実装できます．

(1)ユーザーＩＤの確認

　プッシュ通知機能を使用するには作成したチャネルのユーザーＩＤを確認する必要があります．



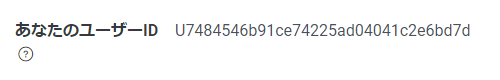


図３．２１　ユーザーＩＤの確認

(3)サンプルプログラムの編集

サンプルプログラム（pushMessage.py）にユーザーＩＤをコピー＆ペーストします．

|  |
| --- |
| #LINEbotAPIインスタンス生成  line\_bot\_api = LineBotApi(channel\_access\_token)  def main():  #ユーザーIDをコピーしてください  user\_id = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  #プッシュ通知  line\_bot\_api.push\_message(  user\_id,  messages=TextSendMessage(text="こんにちは！")  )  if \_\_mane\_\_==’\_\_main\_\_’:  main() |

(4)動作例

サンプルプログラム（pushMessage.py）を実行し，チャネルにメッセージが通知されていることを確認してください．

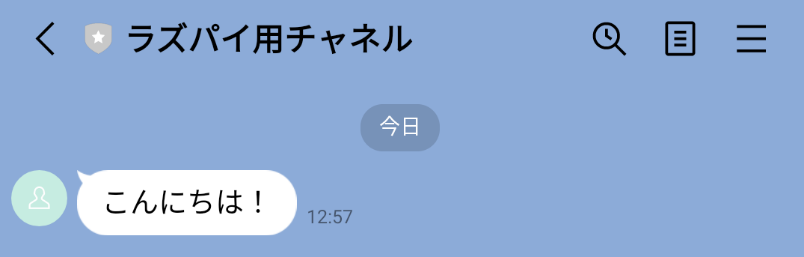


図３．２２　メッセージ通知例

**３．８　練習問題（通知機能を利用した緊急停止の通知）**

(1)通知機能の処理手順

　PLCに実装されているMCプロトコルは基本的にPLC側がサーバであるため，クライアントであるラズベリーパイからコマンドによる問い合わせをすることで稼働状況を取得できます．従ってLINEbotにプッシュ通知機能を実装する際にはポーリングにように一定時間ごとに稼働状況の取得処理をする必要があります．



図２．２３　設備の緊急停止による通知アプリのフローチャート

(2)ファイル名，ファイルパス

ファイル名：EmergencyNotification.py

ファイルパス：/home/pi/work/LINE

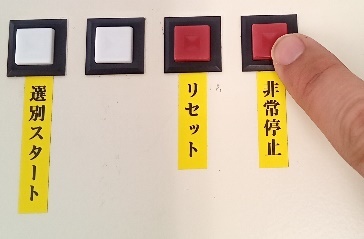
(3)ヒント

|  |
| --- |
| …  #設備の保存用変数  old\_state = “STOP”  while True:  #稼働状況の取得  new\_state = ??????????????()  if ??????????????: #もしold\_stateとnew\_stateが等しくなければ…  if ????????????: #もしnew\_stateが”EMERGENCY”だったら…  #LINE通知  …………  old\_state = new\_state  time\_sleep(1.0) |

(4)動作確認

緊急停止アプリを起動させ，設備の緊急停止ボタンを押したときにLINEに通知がされるか確認しましょう．

|  |
| --- |
| $ python3 EmergencyNotification.py |



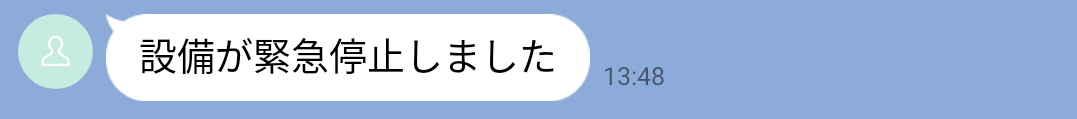


図３．２４　非常停止ボタンを押したことによる通知

(5)追加仕様

　・緊急停止の通知以外に，設備が停止した日付と時刻も同時に通知してください．

　<ヒント>

　Python言語でシステムの日付・時刻を取得するにはdatetimeライブラリをインポートします．

|  |
| --- |
| import datetime  ……  #現在の日付・時刻の取得  #(〇〇〇〇/〇〇/〇〇 〇〇:〇〇)形式(文字列)として取得  days = datetime.datetime.now().strftime(“%Y/%m/%d %H:%M”)  …… |

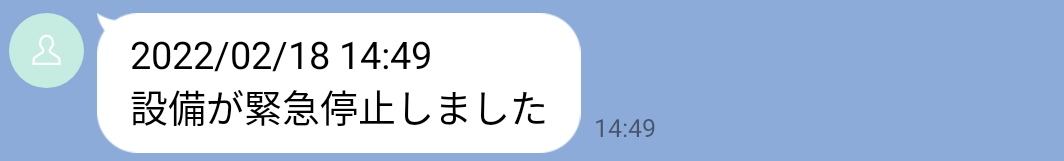


図３．２５　日付・時刻入りの通知

**３．９　いろいろな機能の実装**

**～～ボタンテンプレートによるインターフェースの提供～～**

(1)メッセージイベントの弊害

　LINEbotではユーザーからのメッセージを受信することで様々な動作をさせることができます．しかし，ユーザーは送信するメッセージが自由なため，正しいメッセージを送信しないとLINEbotが動作しないという難点があります．とくに英字の大文字と小文字(例えば”stop”と”STOP”)や同一の意味でも異なる表現(「運転状況」と「稼働状況」)があるため，ちょっとした送信間違いなどが発生しやすいといえます．

　LINEbotにだれがメッセージを送信しても同一の動作をさせるためにLINEbotAPIには**テンプレート**という機能が実装されています．テンプレートには様々なものがありますが今回はボタンテンプレートを紹介します．

(2)テンプレート

　テンプレートとは，文字通りLINEプラットフォームにボタンや画像を表示させメッセージ送信を簡易的に行う機能です．つまり文字を１つ１つタップして打ち込む代わりに「表示されるボタンや画像をタップすることであらかじめ決まったメッセージを送信する」という解釈になります．

　テンプレートには様々なものがあり，それぞれで使用するモジュールが異なります．例としてボタンテンプレートは下記のモジュールをインポートすることで使用できます．

|  |
| --- |
| from linebot.models import (  ……  TemplateSendMessage, ButtonsTemplate, MessageTemplateAction,  ……  ) |

(3)TemplateSendMessageメソッド

　あらかじめ決まったメッセージを送信するためにTemplateSendMessageメソッドを利用します．TemplateSendMessageメソッドには２つの引数があります．

|  |
| --- |
| message = TemplateSendMessage (  alt\_text = “\*\*\*\*”,  template = ……  ) |

alt\_text : 代替テキスト．テンプレートメッセージに非対応のデバイスで表示されるほか、ユーザーがメッセージを受信した際に、端末の通知やトークリストでも表示されます．

template：様々なテンプレート（今回はボタンテンプレートを使用）．ボタンテンプレートを使用するにはTemplateSendMessageメソッドの第二引数であるtemplateにButtonsTemplateメソッドを使用します．

(4) ButtonsTemplateメソッド

　表示されるボタンは下記のようなイメージです．

最初に特定のキーワードを送信



タイトル

メッセージ

アクション

オブジェクト

アクションオブジェクトをタップすることで決まったメッセージが送信される

図３．２６　ボタンテンプレートによる表示

　上記のようなイメージのボタンを表示させるためのButtonsTemplateメソッドは次の引数を指定します．

|  |
| --- |
| message = TemplateSendMessage (  **メッセージ**:選択肢の意味などを説明するために用います  alt\_text = “LED”,  template = ButtonsTemplate (  text = “どちらかを選んでください”,  title= “LED制御”,  actions = [  タイトル：端的でわかりやすいタイトルにします  ……  ]  )  アクションオブジェクト：メッセージや画像がタップした時のアクションを指定する  ) |

(5)メッセージアクション

　アクションオブジェクトはリスト型で指定します．リスト内のアイテムが増えるほどタップできる選択肢（つまりボタンや画像）が増えると考えてください．

　図３．２６のように特定のワードを表示し，その文字をタップすることで決まったメッセージを表示させるアクションのことを**メッセージアクション**といいます．メッセージアクションを使用するにはMessageTemplateActionメソッドを使用します．

|  |
| --- |
| message = TemplateSendMessage (  alt\_text = “LED”,  template = ButtonsTemplate (  text = “どちらかを選んでください”,  title= “LED制御”,  actions = [  MessageTemplateAction (  タップできる表示ラベル  label=”点灯”,  text=”点灯させます！”  表示ラベルをタップすると代わりに送信されるメッセージ  ),  MessageTemplateAction (  label=”消灯”,  text=”消灯させます！”  )  ]  )  MessageTemplateActionメソッド：  1つにつきメッセージ１つ  ) |

注意！：アクションオブジェクトはactions引数にリスト形式で指定します．ただしリストのアイテム数は最大で4件までになります！（つまり1度に表示できるボタンは最大４つまで）

(6)テンプレートの送信

　本来，LINEbotにおけるメッセージはJSON形式で送受信されます(それゆえにPython以外のプログラム言語でもLINEbotAPIが存在します)．TemplateSendMessageメソッドによってJSON形式に変更されますので，戻り値が代入された変数を送信することでテンプレートを送信できます．

|  |
| --- |
| #テンプレートセット  message = TemplateSendMessage (  ……  )  #テンプレート送信  line\_bot\_api.relpy\_message(  event.reply\_token,  message  ) |

(6)生産設備管理の適用事例

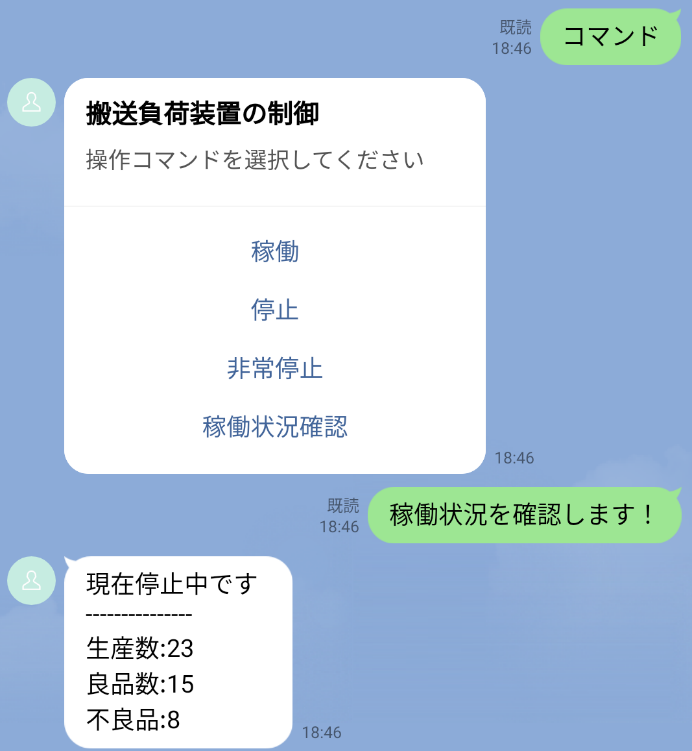


図３．２７　LINEbotを利用した生産設備管理アプリの例

――メモ――

――メモ――