ネットワークモデル　ディベロッパーズガイド
【Ver1.1】

目次

１.BLEを利用してのデータ取得

１－１.ローカルモードへの設定

BLEでデータを取得するためにはピッとシュ本体をローカルモードに設定する必要があります。
ローカルモードには次の手順で設定出来ます。
本体のⓥボタンを長押しすると、本体設定画面になります。「**モード設定**」を選択し、「**ローカルモード**」に設定します。

１－２.BLEアドバタイズについて

ピッとシュネットワーク版はBluetooth4.2をサポートしています。
温度測定後、測定結果はBLEアドバタイズで周囲にブロードキャストします。
アドバタイズは40ms間隔で測定した温度などの情報をアドバタイズします。
各アドバタイズは2秒間で送信を行います。
デバイス名：STPS-002

※BLE通信付加によって、送信間隔がかわることがあるので、ご了承ください。

１.BLEを利用してのデータ取得

１－３.各種アドバタイズデータのフォーマット

アドバタイズパッケージデータフォーマット

アドバタイズデータのフォーマットです。
受信後、シリアル番号を使って新しいデータの判定を行って、BLEアドレスを使ってデータ元を判定できます。
※本プロトコルのすべての数字はリトルエンディアンを採用します。

アドバタイズの長さ	1バイト	0x18
アドバタイズのタイプ	1バイト	0xFF
保留	2バイト	0x0000
シリアル番号	1バイト	0~255，送信データが変わると１増える。255になると元に戻ります。
BLEアドレス	6バイト	BLEアドレス
データパッケージ	14バイト	通信に使用するパッケージは、下記の構成でデータを送信します。 温度データ、消毒データ、NFCカードデータと電池残量データをサポートします。

状態パッケージデータフォーマット

ピッとシュ！が送信必要なデータが無い時、このデータを送信します。
このデータを使ってデバイス接続状況を判断してください。

命令のタイプ	1バイト	0x00
保留	13バイト	

1.BLEを利用してのデータ取得

温度パッケージデータフォーマット

温度測定後、このデータを送信します。

命令のタイプ	1バイト	0x01
温度値	2バイト	符号無整数。単位0.1° 例：36.4℃ ↓ 単位0.1℃ 364 ↓ 16進 0x016C ↓ リトルエンディアン 6C 01
単位	1バイト	0:℃、1:℉
温度状態	1バイト	0：普通，1：警戒，2：高熱 デフォルトの判定範囲は以下になります。 ・普通は32℃～37.5℃ ・警戒は37.5℃～38.5℃ ・高熱は38.5℃～42.9℃ ※ピッとシュ本体で警戒と高熱の範囲を調節できるため、実 際の設置した範囲になります。
UID長さ	1バイト	NFCカード認証後、10秒以内に温度測定する時、温度情報にNFCカードのUIDを一緒に送付する。 UIDの長さは最大8バイト。
UID	Nバイト	

1.BLEを利用してのデータ取得

消毒パッケージデータフォーマト

消毒後、このデータを送信します。

命令のタイプ	1バイト	0x02
液量	1バイト	0：液量少ない，1：残量多い
消毒回数	3バイト	
UID長さ	1バイト	NFCカード認証後、10秒以内に温度測定する時、温度情報にNFCカードのUIDを一緒に送付する。 UIDの長さは最大8バイト。
UID	Nバイト	

カードパッケージデータフォーマット

カードを認証後送信します。

命令のタイプ	1バイト	0x03：新しい情報を読み取る
UID長さ	1バイト	長さ
IDM	8バイト	NFCカードを用いたIDMとなります、足りない桁は0（数字） で埋められます
保留	4バイト	

電池残量パッケージデータフォーマット

電池残量が変わる時送信します。

命令のタイプ	1バイト	0x04
電池電力	1バイト	0~100
保留	12バイト	

2．クラウド上のデータ取得のための設定

2－1.クラウド登録された情報の取得について

クラウドに格納されたデータを取得するには、企業毎に発行される企業コードとパスワード、連携用メールアドレスの登録が必要となります。
下記のアドレスにアクセスし、「**ピッとシュ！データ連携申請**」より申請を行ってください。
https://pitosyu.swiftechie.com/

2－2.アプリのインストール

QRコードを読み取り、端末にアプリをインストールしてください。

App Store

Google Play

2－3．データ連携用アカウントの設定

①ピッとシュ！本体をネットモードに変更します。
　本体のⓋボタンを長押しすると、本体の設定画面になります。「**モード設定**」を選択し「**ネットモード**」に設定して下さい。
②ピッとシュ！アプリの初回起動時画面で、「**データ引継**」をタップします。
　すでにアプリを使用されている場合、再インストールを行うことでこの画面が表示されます。
③メールアドレス入力画面で連携用メールアドレスを入力すると、入力したアドレスに認証コードが届きます。
④認証コード画面で届いた認証コードを入力し、認証が成功すると、測定温度リスト画面になります。
※以降、このアカウントに登録したデバイスはデータ連携用デバイスになり、本資料記載のAPIでデータ取得が出来るようになりますが、
　ピッとシュ！アプリ上でのデータ確認はできなくなります。

初回起動時画面

メールアドレス入力画面

認証コード入力画面

測定リスト画面

2．クラウド上のデータ取得のための設定

2－4.デバイス設定

①測定温度リスト画面で「**追加**」をタップ。デバイス検索が開始されます。
②デバイスが発見されると、デバイス確認画面になります。
③ピッとシュ！が発見されると、デバイス確認画面が表示されます。
④実際に手をかざして温度測定を行います。
　・本体に表示されている温度とアプリ上の温度が一致した場合、「**接続**」をタップ。
　・温度が一致しない場合、「**再検索**」をタップし、デバイスをもう一度検索します。
⑤デバイスとの接続が完了したら、デバイス設定画面が表示されます。
⑥デバイス設定画面でWiFi接続に必要な情報を入力し、確定をタップ。
　デバイス情報の書き換えが行われます。
⑥書き換えが成功すると、測定リスト画面に遷移します。
　以降、本資料記載のAPIでのデータ受信が可能となります。

測定温度リスト画面

デバイス確認画面

自動(ＤＨＣＰ)の場合

手動の場合

デバイス設定画面

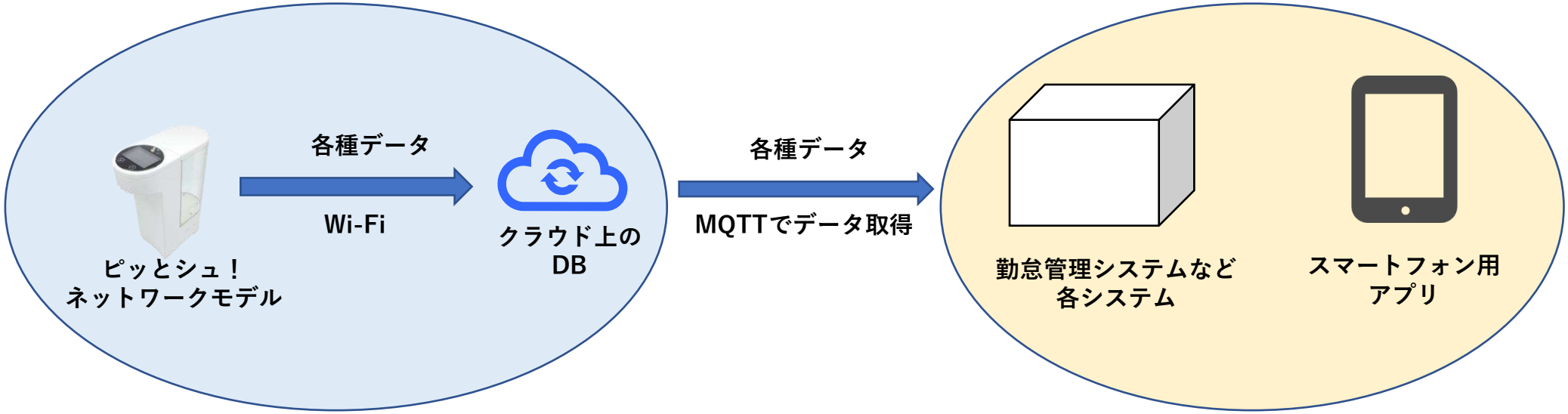
測定温度リスト画面

3.データ提供API

3－1.ご利用いただけるAPIについて

ピッとシュ！クラウドに格納されたデータの取得するには、APIを利用して取得する方法があります。
自社システムに温度情報を取込む方式として、 MQTT_APIの活用し、都度データを取込む方式で自社システム側の開発をお願いします。
都度データの送信については、完全に保証するものではなく、再送要求にも対応しておりません。取得できなかった場合は、ロストデータとして運用をお願いします。
温度データを確実に自社システムとマッチング、管理する要求がある場合は、ネットワークモードではなく、ローカルモードにて近接PCやスマホで、Bluetoothブロードキャストデータを受信して、自社システムに取り込む手順を検討願います。先行ユーザの実績があります。

システムイメージ



弊社提供システム

データを活用したシステムの構築

3.データ提供API

3－2.MQTT_APIについて

・MQTT_APIを利用したpush型情報取得が行えます。

①【資料概要】

デバイスからの検温・消毒情報のプッシュ通知情報を受信する際の内容を定義。
使っているMQTTクライアントの種類により、APIが変わることがあるので、ご注意下さい。

②【基本情報】　受信に用いる基本情報は、下記となります。

項番	項目	内容	備考
1	MQTT バージョン	3.1.1	
2	通信方式	TCP(ポート1883)	
3	プロトコル	mqtt://	
4	ホストURL	pitosyu-mqtt.swiftechie.com	

③【機能一覧】　各機能の詳細は、別シートに記載する。

項番	機能	内容	備考
1	connect	mqttサーバーへ接続する	
2	subscribe	トピックを(subscribe)購読する	
4	unsubscribe	トピックを解除(unsubscribe)する	
5	message push	メッセージが受信した時のコールバック	

3.データ提供API

connect機能

サービス名	mqttサーバー接続
役割	ピッとシュ！クラウドへmqttサーバーを接続する際の内容を定義。

・パラメータ

項番	パラメータ論理名	パラメータ物理名	バリデーション										フォーマット	備考
			必須	最大文字数	文字種									
					半角						全角			
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ		
0	組織ID	client id	○	4	-	○	○	○	-	-	-	-		組織ID
1	ユーザーID	username	○	20	-	○	○	○	-	-	-	-		登録されたメールアドレス
2	パスワード	password	○	20	-	○	○	○	○	-	-	-		パスワード
3	セッション設定	clean session	○	-	-	-	-	-	-	-	-	-	true/false	sessionはサポートしません。必ずtrueを設定
4	接続維持時間	keep alive	○	-	-	○	-	-	-	-	-	-	単位：秒	推薦値：６０分

・戻り値(CONNACK)

項番	戻り値論理名	戻り値物理名	戻り値内容												備考
			必須	最大文字数	文字種								フォーマット	内容	
					半角						全角				
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ			
1	接続結果	returnCode	○	1	-	-	-	-	-	-	-	-	0,1,2,3,4,5のみ	接続の結果を表す	0: connection accepted 1: refused, unacceptable protocal version 2: refused, identifier rejected 3: refused, server unavailable 4: bad user name or pwd 5: refused, not authorized
1	sessionあるか	sessionPresent	○	1	-	-	-	-	-	-	-	-	true/false	sessionがあるかを表す	いつでもfalse

3.データ提供API

connect機能　サンプルソース

・接続リクエストのサンプルソース(kotlin)

```
//接続リクエストを設定
 val opt = MqttClientOptions().setCleanSession(true)
   .setClientId("A001").setUsername("user01@gmail.com").setPassword("my_password")
   .setAutoKeepAlive(true).setKeepAliveInterval(60 * 15)

 //mqttクライアントを立ち上がる
 val client = MqttClient.create(vertx, opt)

 //mqttサーバーを接続
 client.connect(1883, "pitosyu.swiftechie.com") { s ->
   val replyMsg = s.result()
   log.debug("ack code: ${replyMsg.code()}")
   log.debug("is session present? should be false: ${replyMsg.isSessionPresent}")
 }
```

3.データ提供API

subscribe機能

サービス名	トピック購読
役割	トピックを購読する際の内容を定義。

・パラメータ

項番	パラメータ論理名	パラメータ物理名	バリデーション											備考
			必須	最大文字数	文字種								フォーマット	
					半角						全角			
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ		
0	パケットID	packet id			-	○	-	-	○	-	-	-	2 bytes	クライアントよりデフォルト値
1	QoS 1	QoS1		1	-	○	-	-	○	-	-	-	0,1のみ	2をサポートしない
2	トピック 1	topic1		-	-	○	○	○	○	-	-	-	固定	release/d2/v1/{org_id} 購読するトピックは一つのみ

・戻り値(SUBACK)

項番	戻り値論理名	戻り値物理名	戻り値内容												備考
			必須	最大文字数	文字種								フォーマット	内容	
					半角						全角				
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ			
1	パケットID	packet id	○	1	-	-	-	-	-	-	-	-		パケット番号	subscribeリクエストのpacket idと同じ
2	エラーコード	returnCode 1	○	1	-	-	-	-	-	-	-	-		topic1の処理結果	0: success QoS 0 1:success QoS 1 2:success QoS 2 128: failure

subscribe機能　サンプルソース

・購読リクエストのサンプルソース(kotlin)

```
client.subscribe("release/d2/v1/$orgId", 0)
client.subscribeCompletionHandler {
  val id = it.messageId()
  val q = it.grantedQoSLevels()
  log.debug("message id: $id")
  log.debug("QoS granted: $q")
}
```

3.データ提供API

unsubscribe機能

サービス名	トピック解除
役割	トピック購読を解除する際の内容を定義。

・パラメータ

項番	パラメータ論理名	パラメータ物理名	バリデーション											備考
			必須	最大文字数	文字種								フォーマット	
					半角						全角			
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ		
0	パケットID	packet id			-	○	-	-	○	-	-	-	2 bytes	クライアントよりデフォルト値
1	トピック１	topic1		-	-	○	○	○	○	-	-	-	固定	release/d2/v1/{org_id} サポートするのはtopic1のみ

・戻り値(UNSUBACK)

項番	戻り値論理名	戻り値物理名	戻り値内容												備考
			必須	最大文字数	文字種								フォーマット	内容	
					半角						全角				
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ			
1	パケットID	packet id	○	1	-	-	-	-	-	-	-	-		パケット番号	unsubscribeリクエストのpacket idと同じ

unsubscribe機能　サンプルソース

・解除リクエストのサンプルソース(kotlin)

```
client.unsubscribe("release/d2/v1/$orgId")
client.unsubscribeCompletionHandler {
  log.debug("unsub complete. packet id: $it")
}
```

3.データ提供API

message_push機能

サービス名	メッセージプッシュ
役割	ネットワークモデルピットシュでは検温、消毒、残り電量通知など、メッセージをプッシュする処理 メッセージ再配布機能（Durable subscribe）をサポートしないので、各自に処理してください。

・メッセージプッシュ(JSON格式)

項番	論理名	戻り値物理名	戻り値内容											備考
			必須	最大文字数	文字種								フォーマット	内容	
					半角						全角				
					制限なし	数字	アルファベット大文字	アルファベット小文字	記号	カナ	制限なし	カナ			
1	デバイス標識	ble	○	16	-	-	-	-	-	-	-	-	xx:xx:xx:xx:xx	デバイスのブルートゥースのMACアドレス	
2	組織標識	org_id	○	4	-	○	○	○	-	-	-	-			
3	メッセージタイプ	msg_type	○	1	-	○	-	-	-	-	-	-	1,2,3		
3	測定日時	created_at	○	1	-	○	-	-	-	-	-	-	timestamp		UTC timestamps in milliseconds
8	メッセージ内容	payload	○	-	-	○	○	○	○	-	-	-	JSON		

message_push機能　サンプルソース

・プッシュ通知　階層（JSON形式）　　階層のみ表記する。値の内容は前述の戻り値容を参照する。

```
{
  "ble": 値,
  "org_id": 値,
  "created_at": "値"
  "msg_type": "値"
  "payload":  {json}
}
```

・メッセージ サンプル

・温度測定の場合

```
{
  "ble": "50:02:91:91:FB:1G",
  "org_id": orgid,
  "created_at": 1608862044123,
  "msg_type": 1,
  "payload": {  "temperature": 37.1,  "unit": 0,  "status": 0, "card_no": "xxxxxxxx"  }
}
```

・消毒の場合

```
{
  "ble": "50:02:91:91:FB:1G",
  "org_id": orgid,
  "created_at": 1608862044123,
  "msg_type": 2,
  "payload": {  "volueme_left": 37.1,  "spray_total_count": 123, "card_no": "xxxxxxxx" }
}
```

・残り電量の場合

```
{
  "ble": "50:02:91:91:FB:1G",
  "org_id": orgid,
  "created_at": 1608862044123,
  "msg_type": 3,
  "payload": {  "battery": 62 }
}
```

・mqtt受信のサンプルソース(kotlin実現)

```
client.publishHandler{ msg ->
   //メッセージを読み取る
    val json = msg.payload().toJsonObject()
    val payload = json.getJsonObject("paylaod")
    val temperature = payload.getDouble("temperature")
   //後、各自のビジネスロジックを行う
  }
```