

KC4-C-100A

LTE™ Category 4搭載 多機能ゲートウェイ

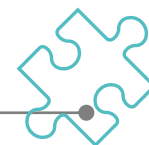
ブロックプログラミング[※]をはじめよう

通信プロトコル編①

Ver.1.0

※ブロックを構成して作るビジュアルプログラミングを
ブロックプログラミングと定義しています。

通信プロトコル編①について

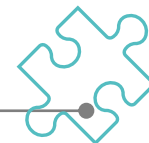


本ドキュメントは「ブロックプログラミングをはじめよう」の「導入編」の続編となります。このためレシピツールのインストールやレシピの機器への書込み等は記載されていないため、最初に「導入編」をご参照ください。



- ・はじめるまでの準備
- ・キッティングツールについて
- ・レシピツールについて
- ・ブロックプログラミングについて
- ・はじめてのブロックプログラミング

■ 通信プロトコル編①について



本編は以下の通信プロトコルの使い方を説明致します。

- ・HTTP
- ・MQTT

※通信プロトコル編②にて以下を説明致します。

- ・HTTPS
- ・MQTTS

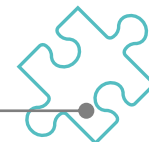
なお、本編ではSIMとクラウドサービスはSORACOMを使用して説明しているため、SORACOM IoT SIMをご準備いただきますと理解が深まります。



■ 通信プロトコルブロックを使用する準備

本章では通信プロトコル関係のブロックを使用する前の準備を説明します。


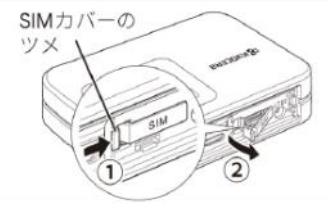
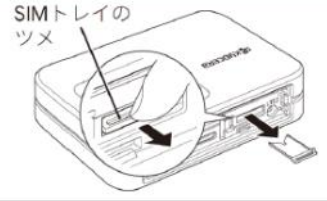
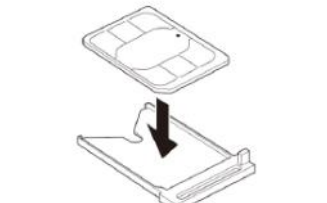
- ・SIMの挿入
 - ・キッティングツールの操作
 - ・APN/認証パラメータの設定
- を行います。

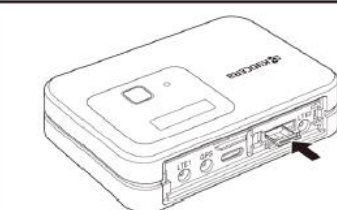
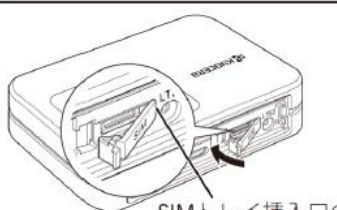
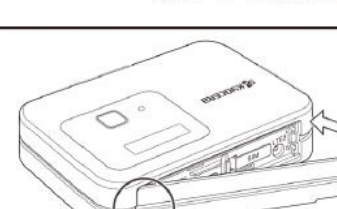


SIMの挿入

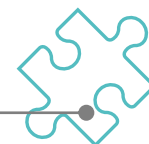
本製品に電源を入れる前にSIMを挿入します。

1.SIMの入れ方

	<p>① 本体カバーの凹みにご自身の指などを引っ掛けてカバーを取り外す ※ 取り外した本体カバーの紛失には気をつけてください。</p>
	<p>② SIMカバーのツメにご自身の爪などを引っ掛けて①の方向に押しながら②の方向へ持ち上げて取り外す ※ 取り外したSIMカバーの紛失には気をつけてください。</p>
	<p>③ SIMトレイのツメに指をかけてまっすぐに引き出す ※ SIMトレイが引き出しにくい場合は、SIMトレイのツメ部分にご自身の爪などを引っ掛けて引き出してください。</p>
	<p>④ SIMトレイに図の向きでnanoSIMカードをのせる ※ nanoSIMカードは金属面を上にしてください。</p>

	<p>⑤ SIMトレイを奥まで挿入する</p>
	<p>⑥ 本体のSIMトレイ挿入口の端に合わせてSIMカバーの先端を挿入し、上から押してしっかりと取り付ける</p>
	<p>⑦ 本体カバーの凸部を本体の穴に合わせて先に挿入し、上から押してしっかりと取り付ける ※ 本体カバーを上からなぞり、本体カバーが浮いていることのないように確実に閉じてください。</p>

※機器の扱いは合わせて同梱の取り扱い説明書をご参照願います。

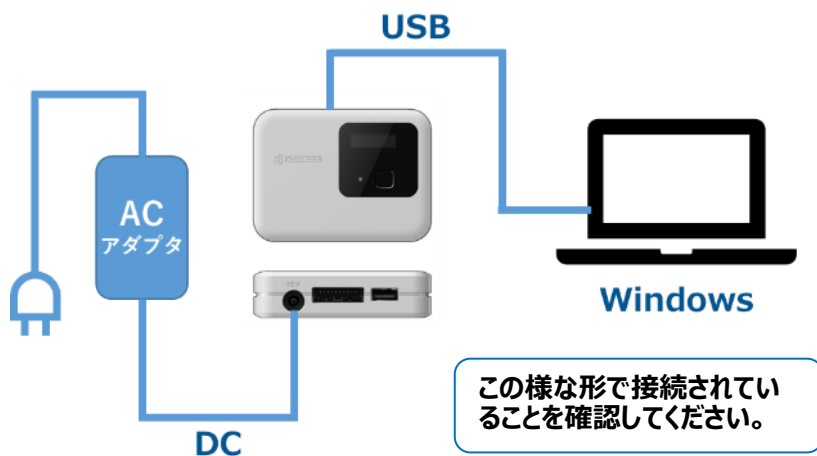


キッティングツールの操作

挿入したSIMの接続先を設定するために、キッティングツールを操作します。

1. キッティングツールの立上げ

キッティングツールの立上げの前に、PCに機器を下記のように接続してください。



レシピツールが正しくインストール完了しているとデスクトップに「K」アイコンが表示されるので、このアプリケーションを実行します。



← このアイコン

2. APNの設定①

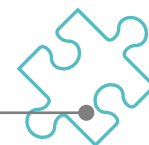
キッティングツールが立ち上がると以下のような画面が表示されます

一覧	ポート	製品名	IMEI	レシビD	認証	操作
COM5	KC4-C-100A		IMEI: E8070116		パスワード未登録	レシピツール起動 レシピツール終了 レシビ更新 設定値読み込み/書き込み パスワード設定 ファームウェア更新 ログ取得

設定項目

APN: FOTA用APN, APN設定

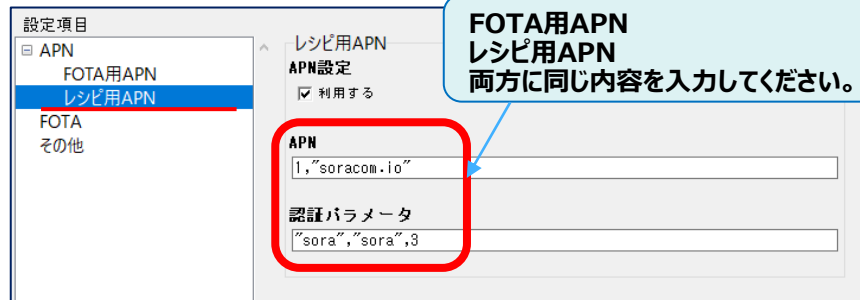
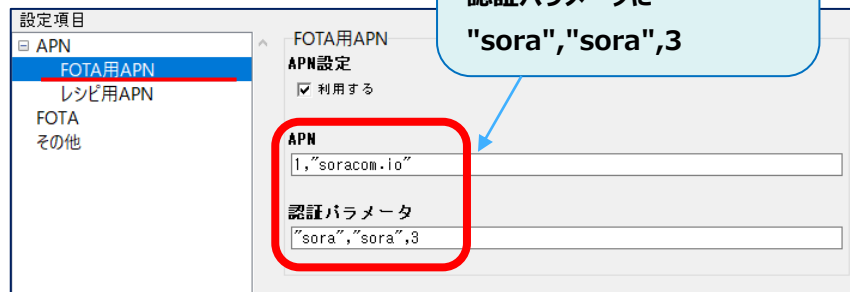
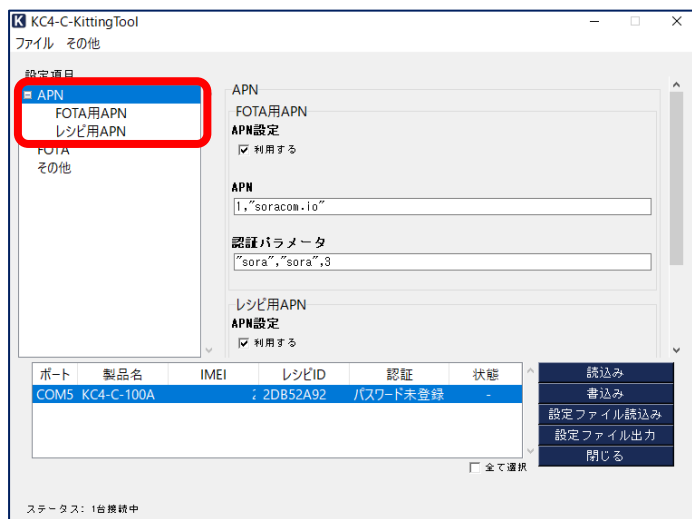
対象の機器を選択して「読み込み」を選択します。



キッティングツールの操作

3.APNの設定②

APNのメニューを選択して、APNの設定を行います。





HTTPブロックの使い方

HTTP Request

Method GET ▾

ホスト名

Uri

ポート番号(0~65535) 80

header

payload i ▾

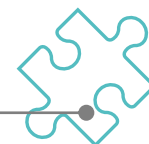
データ送信失敗時 送信データを破棄 ▾

タイムアウト(1~2147483) 7 秒

通信成功

通信失敗

本章ではHTTPブロックの使い方を説明します。



HTTPブロックの機能説明

HTTP Requestブロックの要素を以下に説明します。

HTTPブロック

HTTP Request

Method GET

ホスト名

Uri

ポート番号(0~65535) 80

header

payload i

データ送信失敗時 送信データを破棄

タイムアウト(1~2147483) 7 秒

通信成功

通信失敗

Method

Method GET

GET

POST

ポート番号(0~65535) 80

MethodにはGET/POSTが選択出来ます。
GETの場合は以下のブロックでResponse Dataを取得出来ます。

文字変数 k に Response Data をセット

マップ変数 m に Response Data をセット

ホスト名,Uri,port

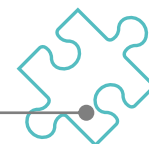
ホスト名 " beam.soracom.io "

Uri "/test/"

ポート番号(0~65535) 80

ホスト名, Uriには文字列または文字変数を接続し、
Portにはport番号を数字で入力します。

上図の例では、URL表記をすると、
http://beam.soracom.io:80/test/ になります。



HTTPブロックの機能説明

Header

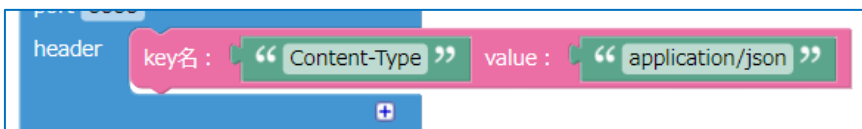
httpのheaderを指定します。



+を押すとHeaderを入力するmap変数が生成されます。



例えばContent-Typeを指定する場合は、以下のようになります。

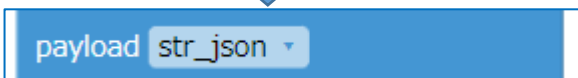
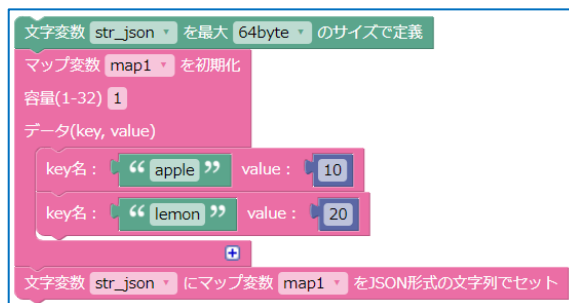


Payload

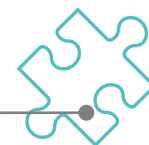
Payloadには文字変数を指定します。



例えば、下記の様にJson形式の文字列を準備して、payloadに設定します。



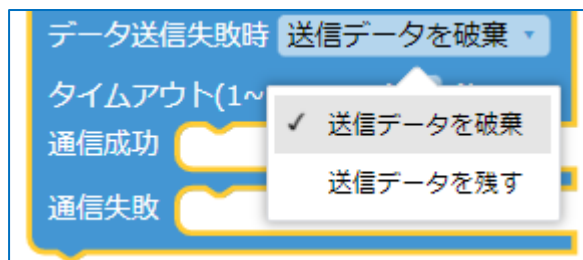
→{"apple":10,"lemon":20}



HTTPブロックの機能説明

データ送信失敗時

データ送信失敗時の振舞いを設定出来ます。



送信データを破棄

圏外時などの送信失敗時にHTTPパケットは破棄されます。

送信データを残す

送信失敗したHTTPパケットは保持され、次のHTTP requestが発行された際に送信されます。

※「送信データを残す」は現バージョンでは不備がありますので、「送信データを破棄」でご使用ください。データを残す機能は今後のアップデートで対応致します。

タイムアウト

サーバー応答までのタイムアウト時間を設定出来ます。

タイムアウト(1~2147483) 7 秒

通信成功/失敗

通信成功/失敗時の処理を設定することが出来ます。

通信成功

通信失敗

通信の結果のStatusを取得するには以下のブロック

数値変数 `j` に Return Status Code をセット

応答のデータを取得するには以下のブロック

文字変数 `k` に Response Data をセット

マップ変数 `m` に Response Data をセット

※現行のバージョンでは送信失敗時のReturn Status CodeやResponse Dataはセット機能していない為、今後のアップデートで対応致します。

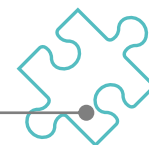


HTTPブロックのチュートリアル

本章ではHTTPブロックの使い方を演習します。
 演習レシピとしては、

- GPSデータ取得
- 現在時刻の取得
- Jsonデータの作成
- HTTP送信

を行います。



チュートリアル① GPSの値と現在時刻を定期的にサーバに送信します。

1. レシピの新規作成

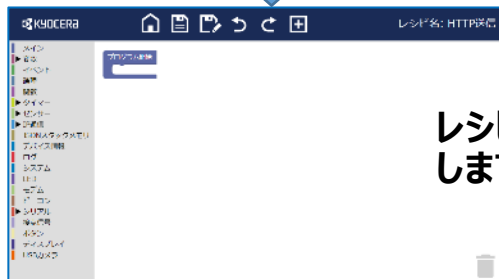
新しいレシピの作成には新規作成を押下します。



レシピの新規作成



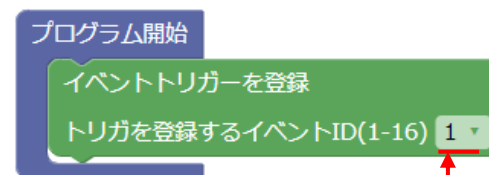
タイトルを入力して新規作成ボタンを押下します。(説明は任意です。)



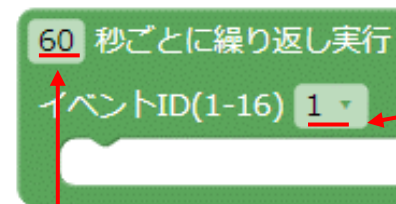
レシピ編集画面に遷移します。

2. イベントトリガー/ブロックの配置

「プログラム開始」に「イベントトリガーブロック」をドラッグして接続します。

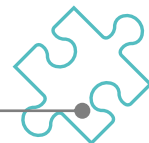


以下のイベントブロックを配置します。



イベント番号が同じことを確認ください。

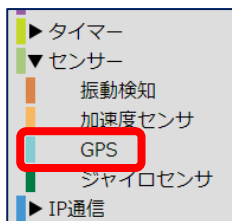
今回は60秒ごとに処理を行いますので、60秒に変更します。



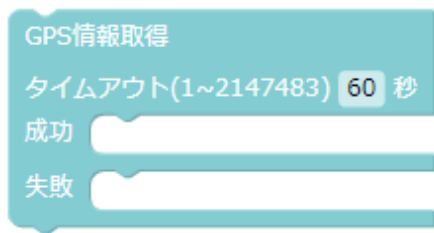
チュートリアル②

4. GPSブロックの配置

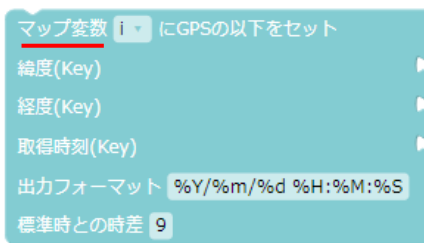
センサーカテゴリの中にあるGPSブロックを配置します。



GPSは取得するためのブロックと



取得したGPSの値と現在時刻を保存するブロックが準備されています。



保存する先がマップ変数になります。

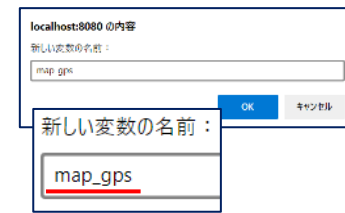
上記2つのブロックを配置します。

5. マップ変数を作成

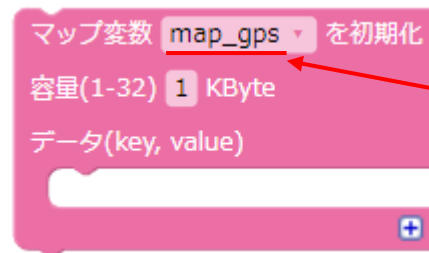
取得した値を格納するマップ変数という入れ物を作ります。



map_gpsと言う変数名を作ります

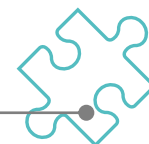


マップ変数を初期化するブロックを配置し、変数名を変更します。



上記で作成した変数名を選択します。

※マップ変数とは、keyとvalueで構成される2元配列変数のイメージです。

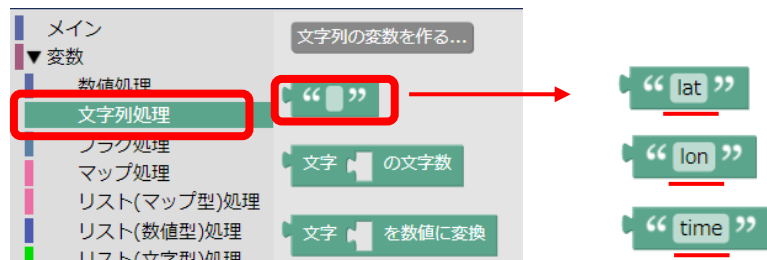


チュートリアル③

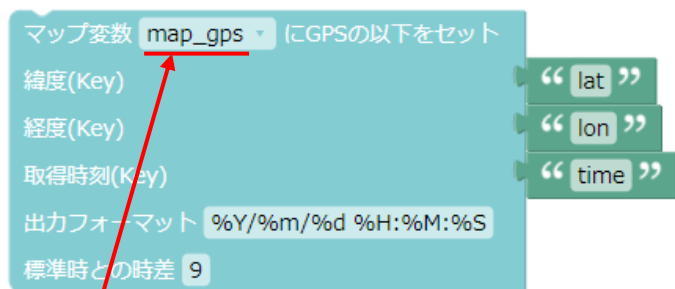
6. JsonのKey名の作成

送信するJsonのKey名を作成します。

「文字列処理」の「固定文字列」ブロックを3つ作り、文字列として lat, lon, time を入力します。



固定文字列をGPS取得ブロックに接続していきます。

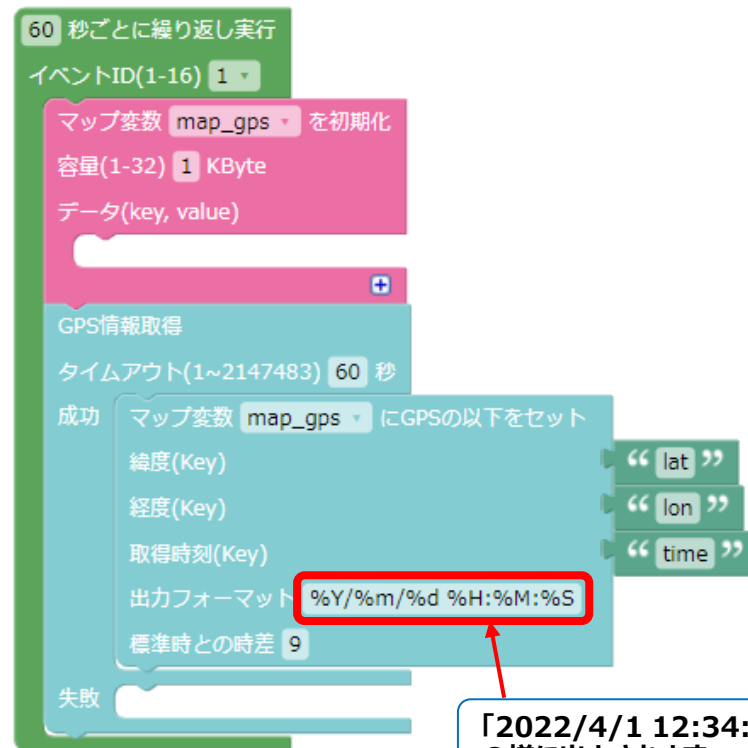


map変数名も変更します。

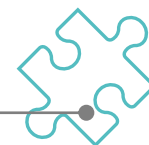
7. GPSブロックの組み立て

GPSとマップ変数のブロックを組み立てます。

イベントブロックとも合わせて以下のようになります。



「2022/4/1 12:34:56」
の様に出力されます



チュートリアル④

8.HTTPブロックの配置

本題であるHTTPブロックを配置します。

POSTに変更します。

「固定文字列」でホスト名とUriを作り接続します。

Uriは空で問題ありません

9.HTTPヘッダの作成

HTTPのヘッダを指定します。

Headerを追加するには下記「+」を押します。

↓ 入力ブロックが現れます

↓ ここに「固定文字列」として、Content-Type, application/jsonを追加します。



チュートリアル⑥

12.HTTPのpayloadの指定

作成したJson文字列をHTTPブロックへ結合し、payloadに指定します。

文字変数定義とJson変換ブロックを結合

文字変数 `str_json` を最大 1280byte のサイズで定義
文字変数 `str_json` にマップ変数 `map_gps` をJSON形式の文字列でセット

HTTP Request
Method POST
ホスト名 uni.soracom.io
Uri /
ポート番号(0~65535) 80
header key名: Content-Type value: application/json
payload str_json
データ送信失敗時 送信データを破棄
タイムアウト(1~2147483) 7 秒
通信成功
通信失敗

※データ送信失敗時の設定は「送信データを破棄」でご使用ください。

Payloadにstr_jsonを指定

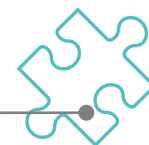
13.GSPブロックとHTTPブロックの結合

前記までのブロックを結合すると以下様になります。

60 秒ごとに繰り返し実行
イベントID(1-16) 1
マップ変数 map_gps を初期化
容量(1-32) 1 KByte
データ(key, value)
GPS情報取得
タイムアウト(1~2147483) 60 秒
成功 マップ変数 map_gps にGPSの以下をセット
緯度(Key) lat
経度(Key) lon
取得時刻(Key) time
出力フォーマット %Y/%m/%d %H:%M:%S
標準時との時差 9
失敗
文字変数 map2json を最大 1280byte のサイズで定義
文字変数 map2json にマップ変数 map_gps をJSON形式の文字列でセット
HTTP Request
Method POST
ホスト名 uni.soracom.io
Uri /
ポート番号(0~65535) 8888
header key名: Content-Type value: application/json
payload map2json
データ送信失敗時 送信データを破棄
タイムアウト(1~2147483) 7 秒
通信成功 ディスプレイへ文字を表示 SEND_SUCCESS
通信失敗 ディスプレイへ文字を表示 SEND_FAIL

GPS関連のブロック

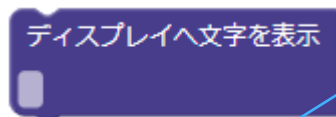
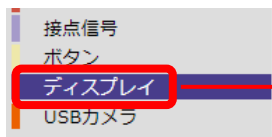
HTTP関連のブロック



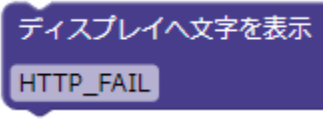
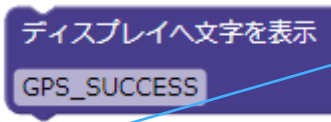
チュートリアル⑦

14.ディスプレイブロックの配置

前記までのブロックでも動作をしますが、端末の動作状態がわからないため、状態の表示を追加します。



ここでは以下の5つの状態表示を作成し、処理の各場所へ配置を行います。



```
60 秒ごとに繰り返し実行
イベントID(1-16) 1
ディスプレイへ文字を表示
GPS_SCCANNING
マップ変数 [map_gps] を初期化
容量(1-32) 1 KByte
データ(key, value)
GPS情報取得
タイムアウト(1~2147483) 60 秒
成功 マップ変数 [map_gps] にGPSの以下を設定
緯度(Key) "lat"
経度(Key) "lon"
取得時刻(Key) "time"
出力フォーマット %Y/%m/%d %H:%M:%S
ディスプレイへ文字を表示
GPS_SUCCESS
失敗 デisplayへ文字を表示
GPS_FAIL
文字変数 [map2json] を最大 [1280byte] のサイズで定義
文字変数 [map2json] にマップ変数 [map_gps] をJSON形式の文字列でセット
HTTP Request
Method POST
ホスト名 "uni.soracom.io"
Uri ""
ポート番号(0~65535) 8888
header key名: "Content-Type" value: "application/json"
payload [map2json]
データ送信失敗時 [送信データを破棄]
タイムアウト(1~2147483) 7 秒
通信成功 デisplayへ文字を表示
SEND_SUCCESS
通信失敗 デisplayへ文字を表示
SEND_FAIL
```

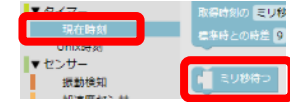
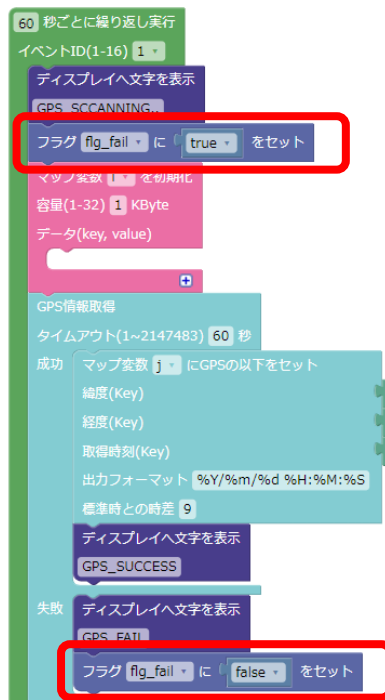
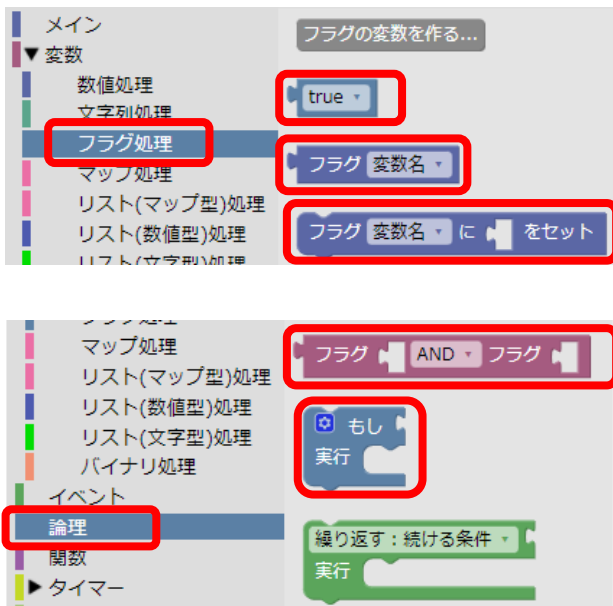


チュートリアル⑧

15.送信条件の追加

現状の動作では、GPS捕捉が失敗した場合においてもデータの送信を行ってしまうため、成功した場合のみ送信するよう条件を設定します。

ここでは「フラグ変数」と「論理」を使用して構成しています。下記ブロックを用いて、先のようにブロックを構成してください。

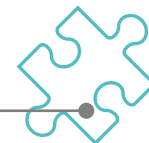


※表示の変化が見やすいように1秒のウェイトを置いてます。「タイマー」「現在時刻」の中にあるブロックです。



※ブロックの構成は右上の矢印に続きます。

HTTP関連ブロックは「もし」ブロックの実行の中に配置します。



チュートリアル⑨

16.レシピの完成

前記完成したレシピは右図のようになります。次からレシピ実行ファイルの作成を行いますので、変数名等の間違いがないか、確認してください。



※完成したレシピの保存を忘れないようにお願いします。

プログラム開始

- イベントトリガーを登録
 - トリガを登録するイベントID(1-16) 1

60 秒ごとに繰り返し実行

- イベントID(1-16) 1
- ディスプレイへ文字を表示
 - GPS_SCCANNING..
- フラグ `flag_fail` に `true` をセット
- マップ変数 `map_gps` を初期化
 - 容量(1-32) 1 KByte
 - データ(key, value)
- GPS情報取得
 - タイムアウト(1~2147483) 60 秒
- 成功
 - マップ変数 `map_gps` にGPSの以下をセット
 - 緯度(Key) `"lat"`
 - 経度(Key) `"lon"`
 - 取得時刻(Key) `"time"`
 - 出力フォーマット `">%Y/%m/%d %H:%M:%S`
 - 標準時との時差 9
 - ディスプレイへ文字を表示
 - GPS_SUCCESS
- 失敗
 - ディスプレイへ文字を表示
 - GPS_FAIL
 - フラグ `flag_fail` に `false` をセット

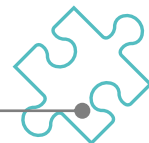
※ブロックの構成は右上の矢印に続きます。

1000 ミリ秒待つ

もし フラグ `flag_fail` AND フラグ `true`

実行

- 文字変数 `str_json` を最大 1280byte のサイズで定義
- 文字変数 `str_json` にマップ変数 `map_gps` をJSON形式の文字列でセット
- HTTP Request
 - Method POST
 - ホスト名 `"uni.soracom.io"`
 - Uri `"`
 - ポート番号(0~65535) 80
 - header
 - key名: `"Content-Type"` value: `"application/json"`
 - payload `str_json`
 - データ送信失敗時 送信データを破壊
 - タイムアウト(1~2147483) 7 秒
 - 通信成功
 - ディスプレイへ文字を表示
 - HTTP_SUCCESS
 - 通信失敗
 - ディスプレイへ文字を表示
 - HTTP_FAIL



チュートリアル⑩

17.レシピの機器へのダウンロード

レシピ実行ファイルの作成と機器へのダウンロード方法は、「導入編」をご参照ください。



導入編



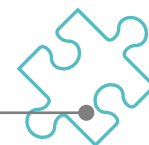
レシピ実行ファイルを機器へダウンロードする手順ページ参照

18.機器の動作

レシピをダウンロードすると、機器上でレシピが動作を開始し、GPSを捕捉に行き、以下のような画面遷移になります。



※GPSのスキャンが長い場合、表示の焼付きを防止するために、10秒程度で画面が消灯します。



チュートリアル⑪

19.送信データの確認①

送信データを確認するための設定を行います。

1-1.ログイン

SORACOMコンソールのログイン

以下のURLからログインします。

<https://console.soracom.io>



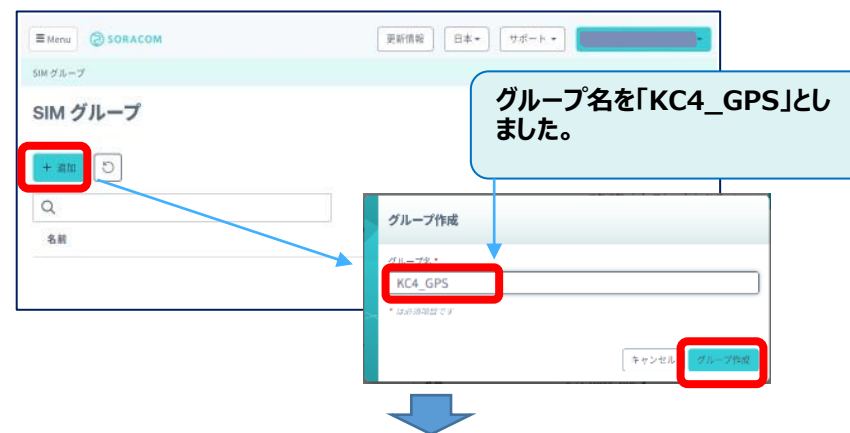
1-2.グループの作成①

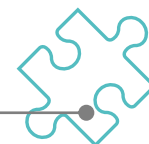
SORACOMコンソールにログインをし、左上の「Menu」を選択し、その中の「SIMグループ」を選択します。



1-3.グループ作成②

左にある「+追加」ボタンをクリックします。



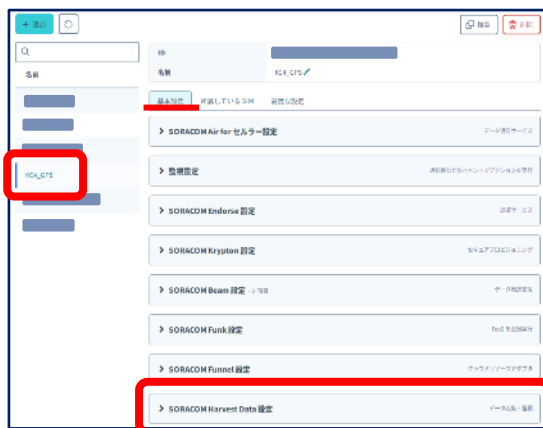


チュートリアル⑫

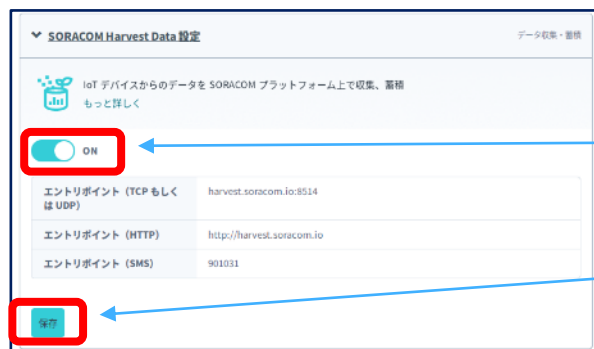
20.送信データの確認②

2-1. SORACOM Harvest Dataを設定

「基本設定」の中のSORACOM Harvest Dataを開いて「ON」にします。



SORACOM Harvest Data



ON

保存

2-2. 設定確認

費用が発生する注意事項を確認の上、「OK」を押下します。

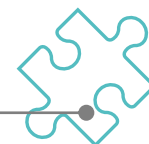


※データのリクエスト数に伴い、利用料が発生します。

「SORACOM」ロゴをクリックしてコンソール画面に戻ります。



グループ設定が完了したら
コンソール画面に戻ります。



チュートリアル⑬

2-3.SIMへのSIMグループの適用

機器に挿入した該当のSIMにSIMグループを適用します。

このスクリーンショットは、SIM管理画面の「SIM登録」タブに移動した状態を示しています。検索バーには「KC4-C-100A」と入力されています。SIM一覧表の最初の行「KC4-C-100A」の「詳細」ボタンが赤い枠で囲まれています。また、このSIMのチェックボックスも赤い枠で囲まれています。

該当のSIMを選択して、「詳細」をクリックします。

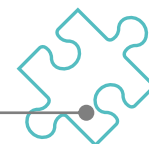


このスクリーンショットは、SIM「KC4-C-100A」の詳細ページです。SIM ID、IMSI、名前（KC4-C-100A）、グループ（KC4_GPS）などの情報が表示されています。グループのプルダウンメニューが赤い枠で囲まれています。また、「閉じる」ボタンも赤い枠で囲まれています。

先程作成したグループを選択します。

該当のSIMにグループが設定されたことを確認します。

このスクリーンショットは、SIM管理画面の「SIM登録」タブに戻った状態を示しています。検索バーには「KC4-C-100A」と入力されています。SIM一覧表の最初の行「KC4-C-100A」の「グループ」列に「KC4_GPS」が設定されていることが確認できます。この「KC4_GPS」の文字も赤い枠で囲まれています。



チュートリアル⑭

21. 送信データの確認

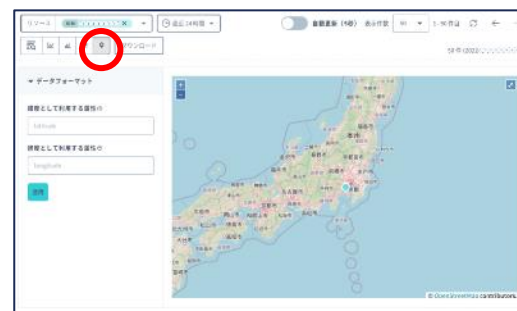
送信したデータを確認します。



対象のSIMにチェックを入れて、
「操作」-「データ確認」

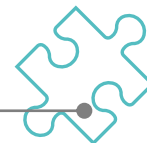


送信されたGPS情報が表示されます。



マップ表示にすると地図上で位置が表示されます。

チュートリアル⑫



22. HTTPブロックチュートリアルの終了

以上でHTTPのチュートリアルを終了します。

お疲れさまでした。

通信プロトコルのブロックプログラミングは少し複雑に感じることもありますが、慣れてしまえば今回のレシピも数分で組むことができます。

他にもいろんな機能ブロックがありますので、オリジナルレシピの作成にトライしてみてください。





MQTTブロックの使い方

本章ではMQTTブロックの使い方を説明します。

MQTTブローカーとクライアントID

サーバアドレス

ポート番号(0~65535)

keepalive(0~65535) 秒

Clean Session

QoS

ユーザー名

パスワード

タイムアウト(1~2147483) 秒

ブローカー接続成功

ブローカー接続失敗

last will topic

last will message

トピックにメッセージをパブリッシュ

トピック

メッセージ

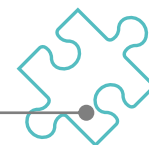
データ送信失敗時

タイムアウト(1~2147483) 秒

成功

失敗

MQTT接続を切断



MQTTブロックの機能説明

MQTTとパブリッシュブロックの要素を以下に説明します。

MQTTブロック

MQTTブローカーとクライアントID

サーバアドレス

ポート番号(0~65535)

keepalive(0~65535) 秒

Clean Session

QoS

ユーザー名

パスワード

タイムアウト(1~2147483) 秒

ブローカー接続成功

ブローカー接続失敗

last will topic

last will message

パブリッシュブロック

トピックにメッセージをパブリッシュ

トピック

メッセージ

データ送信失敗時

タイムアウト(1~2147483) 秒

成功

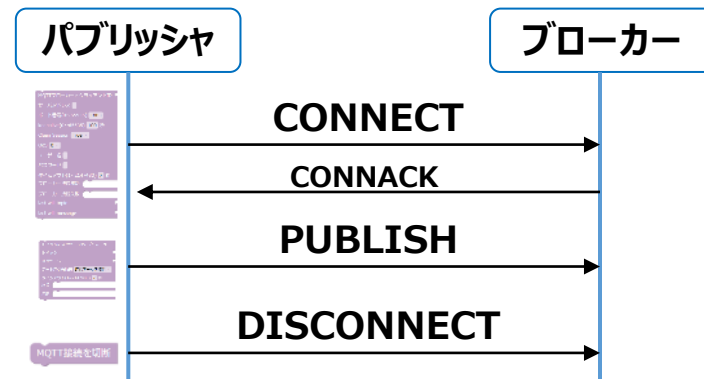
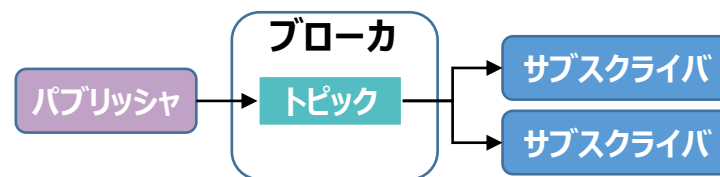
失敗

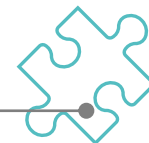
切断ブロック

MQTT接続を切断

MQTTプロトコル

本機器のMQTTはV3.1.1相当となり、機能はパブリッシャのみ対応しています。





MQTTブロックの機能説明

MQTTブロック説明

クライアントID

クライアントIDを文字列で指定します。

MQTTブローカーとクライアントID `“client1”`

※クライアントIDを空文字 `“ ”` にすることも可能ですがこの場合はサーバー側で任意文字が割り振られる場合の他、エラーとなる場合があります。

(また、クライアントから同じSessionへの接続は出来なくなります。)

サーバアドレス/ポート番号

接続するサーバのアドレスとポート番号を指定します。

サーバアドレス `beam.soracom.io`

ポート番号(0~65535) `1883`

※テスト用に公開されている「test.mosquito.org」などのブローカーでも接続確認は出来ます。

MQTTのポート番号は一般的に1883が使われますがサーバーの指定に従ってください。

Keepalive

Sessionの継続時間を指定します。

keepalive(0~65535) `100` 秒

※この時間の間に同クライアントIDでの接続か、切断処理をしない場合、サブスクライバにWill(遺言)が発行されます。

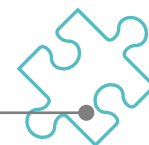
Clean Session

既存Sessionを破棄するかどうかを指定します。

Clean Session `True`
QoS `0`
ユーザー `False`

True: 既存セッションを破棄し新しいセッションで接続します。
False: 既存のセッションで接続します。

※Trueで接続した場合、事前にQoS=1で送信したメッセージをサブスクライバが受け取って無い時に、ブローカー側でそのメッセージがクリアされます。



MQTTブロックの機能説明

MQTTブロック説明

QoS

Quality of Serviceを指定します。



- QoS0 : メッセージの到着確認は行わない。
- QoS1 : メッセージの到着確認を行う。(パブリッシャー-ブローカー間)
- QoS2 : サブスクライバまでの到着確認を行う。
(チュートリアルでのAWSはQoS1までの対応となります。)

ユーザー名/パスワード

ユーザー名/パスワード認証を行うサーバに対しての設定を行います。

ユーザー名

パスワード

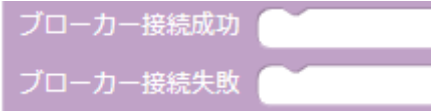
タイムアウト

ブローカー応答までのタイムアウト時間を設定します。

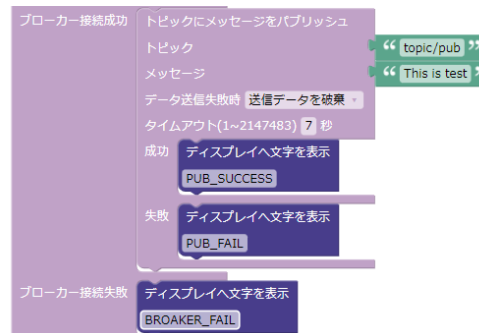
タイムアウト(1~2147483) 7 秒

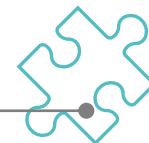
通信成功/失敗

ブローカーへの通信成功/失敗時の処理を設定することが出来ます。



通常は以下の様にこの中にパブリッシュブロックを配置します。





MQTTブロックの機能説明

MQTTブロック説明

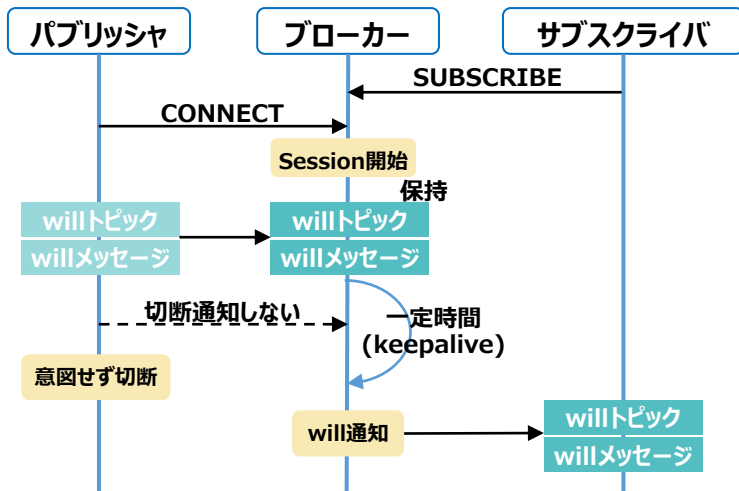
パブリッシュブロック説明

Last will

willトピックとメッセージを指定します。

last will topic
last will message

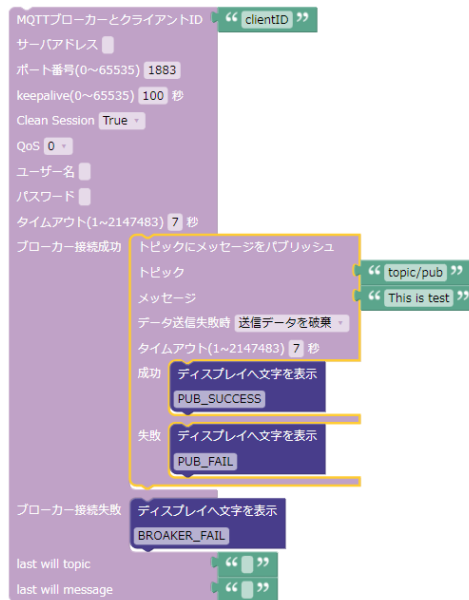
willはパブリッシャが意図せずに切断された場合等に、サブスクライバに通知を出す機能となります。パブリッシャ側が正しく切断した場合は通知は出力されません。

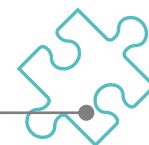


パブリッシュブロック

パブリッシュブロックはブローカー接続成功の中に配置して、対象トピックにメッセージを送信します。

例としては下記のように「ブローカー接続成功」の中にパブリッシュブロックを配置してメッセージを送信します。





MQTTブロックの機能説明 パブリッシュブロック説明

トピックとメッセージ

パブリッシュするトピックとメッセージを指定します。



上記の例では、「topic/pub」の階層構造の中に「This is test」メッセージを送信する形になります。

実際には、Json形式のメッセージなどを送信する形になるので、ブロックとして前段でJson形式の文字列を作成して接続します。

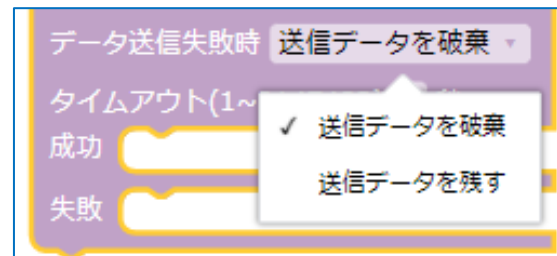


Json文字列の作成

topic/の下にJsonのメッセージ送信する様に指定

データ送信失敗時

データ送信失敗時の振舞いを設定出来ます。



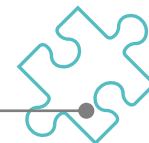
送信データを破棄

送信失敗時にトピックとメッセージのパケットは破棄されます。

送信データを残す

送信失敗したトピックとメッセージのパケットは保持され、次のパブリッシュリクエストが発行された際に送信されます。

※「送信データを残す」は現バージョンでは不備がありますので、「送信データを破棄」でご使用ください。データを残す機能は今後のアップデートで対応致します。



MQTTブロックの機能説明

パブリッシュブロック説明

タイムアウト

サーバー応答までのタイムアウト時間を設定出来ます。

タイムアウト(1~2147483) 7 秒

通信成功/失敗

パブリッシュ成功/失敗時の処理の設定が出来ます。

成功

失敗

切断ブロック説明

MQTT接続を切断

ブローカーに対して切断命令を発行します。

MQTT接続を切断

Last willの説明のところにも記載がありますが、接続したセッションは明示的に切断しないと、サブスクライバにwill通知が発行されます。

単にデータを送信してサーバーへ集めるなどのケースでは、以下の設定で必ずパブリッシュ後に切断する処理が良いと考えます。

MQTTブローカーとクライアントID ClientID 任意

サーバーアドレス beam.soracom.io

ポート番号(0-65535) 1883

keepalive(0-65535) 100 秒

Clean Session True

QoS 0 or 1

ユーザー名

パスワード

タイムアウト(1~2147483) 7 秒

トピックにメッセージをパブリッシュ

トピック

メッセージ

データ送信失敗時 送信データを放棄

タイムアウト(1~2147483) 7 秒

成功

失敗

MQTT接続を切断

パブリッシュ後に必ず切断

トピック

文字列 map_to_json

ブローカー接続失敗

last will topic

last will message

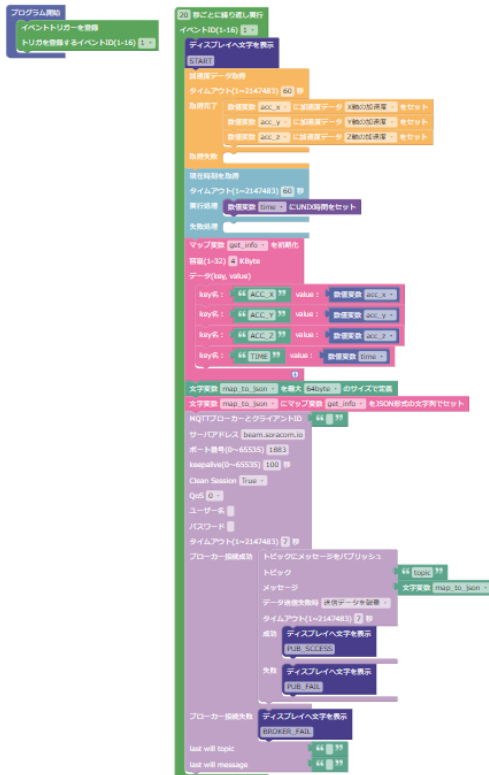


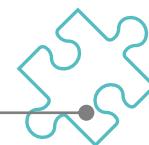
MQTTブロックのチュートリアル

本章ではMQTTブロックの使い方を演習します。
 演習レシピとしては、

- ・加速度の傾きデータ、現在時刻を取得
- ・Json形式でMQTT送信を行います。

※本チュートリアルではSORACOM IoT SIMに加えて、AWSも利用しますので、AWSのアカウントが必要となります。



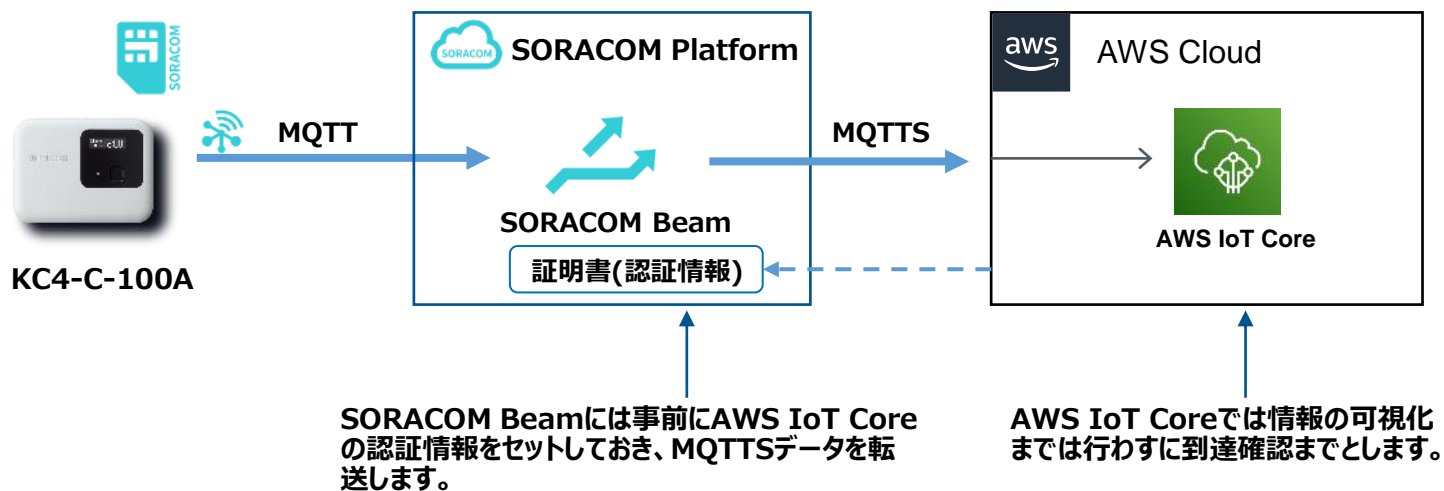


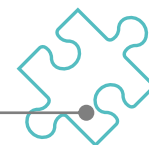
チュートリアル①

1. データの流れ

本チュートリアルでは、2つのクラウドサービスを利用します。

1. SORACOM (IoT SIMとSORACOM Beamの利用)
2. AWS (MQTTS受信で利用)





チュートリアル②

1.AWSでの受信準備

AWS IoT CoreでMQTTの packets を受信できるように準備を行います。

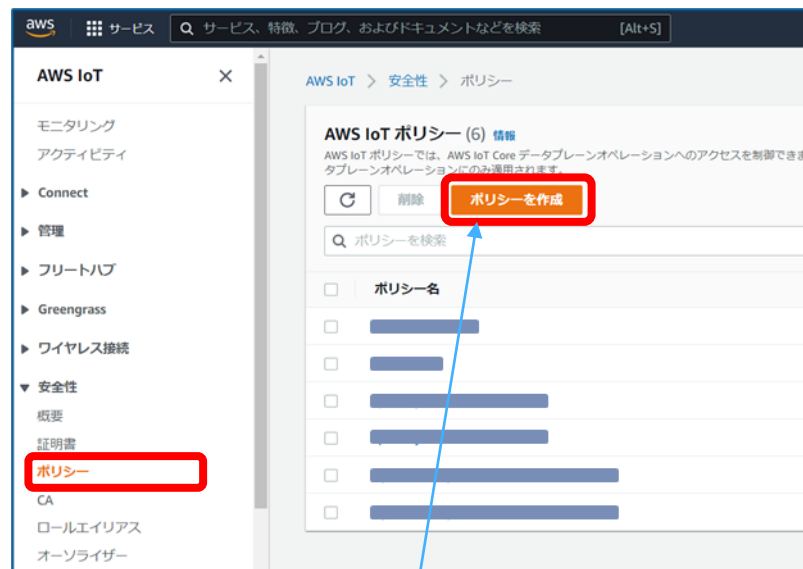
AWSコンソールにログインし、左上の「サービス」を選択して、その中の「IoT」を選択し、右に出るメニューの「IoT Core」を選択します。



1-1.ポリシーの作成①

まず、ポリシーを作成します。

AWS IoTメニューの「安全性」の「ポリシー」を選択すると、右側にポリシー一覧が現れます。
(※初めての場合はポリシーのリスト空となります。)



「ポリシーを作成」ボタンをクリックします。



チュートリアル③

1-2.ポリシーの作成②

ポリシー名、ポリシーアクション、ポリシーリソースの入力/選択を行い、「作成」をクリックします。

The screenshot shows the 'Create Policy' page in the AWS IoT console. The 'Policy Name' field contains 'KC4-policy'. The 'Policy Actions' section has 'iot:Connect' and 'iot:Publish' selected. The 'Policy Resources' section has '*' entered in both fields. A 'New Statement' button is highlighted with a red dashed box. The 'Create' button is also highlighted with a red box.

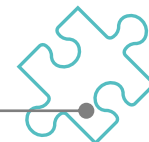
Annotations:

- 名前は任意です。ここでは「KC4-policy」にしました。
- ポリシーアクションは、「iot:Connect」「iot:Publish」を選択します。
- ポリシーリソースは、「*」「*」を入力します。
- ポリシードキュメントの行を増やす為に「追加」をします。

1-3.ポリシーの作成③

ポリシー作成が完了すると、以下のように正常に作成されましたの表示と、該当のポリシーが表示されます。

The screenshot shows the 'Policy List' page in the AWS IoT console. A green notification banner at the top says 'Policy KC4-policy was created successfully'. The policy list below shows 'KC4-policy' selected and highlighted with a red box.



チュートリアル④

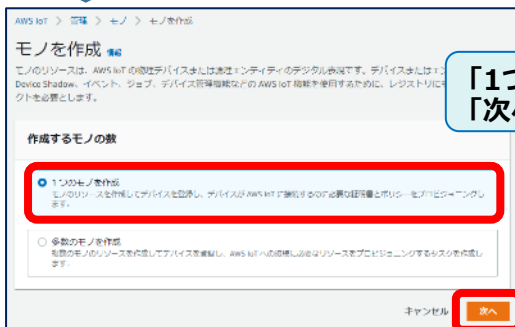
1-4.モノの作成①

AWS IoTメニューの「管理」の「モノ」を選択すると、右側にモノの一覧が現れます。

(※初めての場合はモノのリストは空となります。)



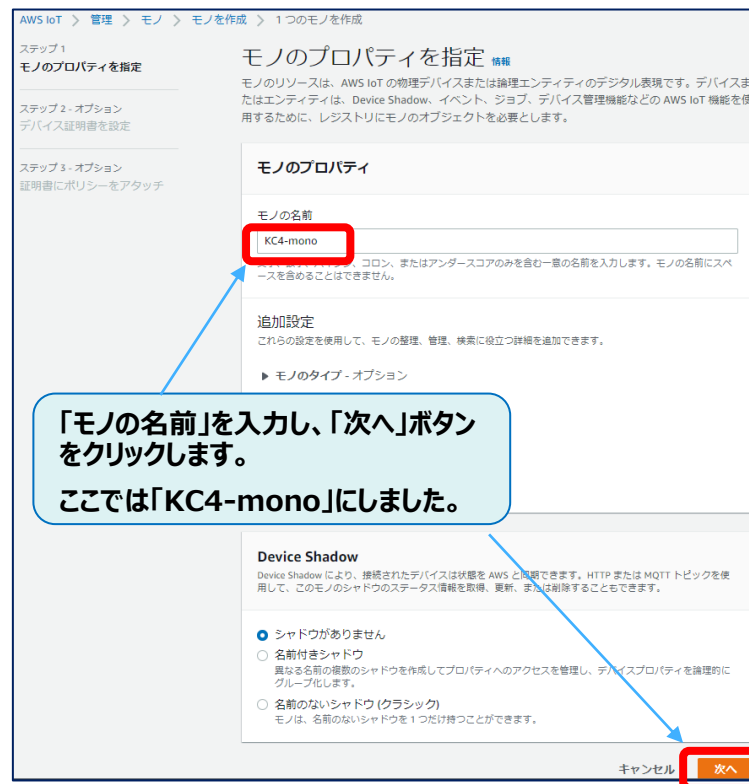
「モノを作成」ボタンをクリックします。



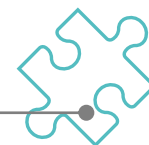
「1つのモノを作成」を選択し「次へ」ボタンをクリックします。

1-5.モノの作成②

「モノの名前」を入力してモノの作成をします。追加設定等は特に変更しません。



「モノの名前」を入力し、「次へ」ボタンをクリックします。
ここでは「KC4-mono」にしました。



チュートリアル⑤

1-6.デバイス証明書の作成

モノの作成の流れで「デバイス証明書」の作成を行います。

AWS IoT > 管理 > モノ > モノを作成 > 1つのモノを作成

ステップ1
モノのプロパティを指定

ステップ2 - オプション
デバイス証明書を設定

ステップ3 - オプション
証明書にポリシーをアタッチ

デバイス証明書を設定 - オプション

デバイスには、AWS IoT に接続するために証明書が必要です。今すぐデバイスの証明書を登録する方法を選択するか、後でデバイス用の証明書を作成して登録できます。適切なポリシーを含むアクティブな証明書がないと、デバイスから AWS IoT に接続することはできません。

デバイス証明書

- 新しい証明書を自動生成 (推奨)
AWS IoT の認証機能を使用して、証明書、パブリックキー、およびプライベートキーを生成します。
- 自分の証明書を使用
独自の認証機能によって署名された証明書を使用します。
- CSR をアップロード
CA を登録し、1 つまたは複数のデバイスに独自の証明書を 사용합니다。
- 今回の証明書の作成をスキップ
このモノの証明書を作成し、後で証明書にポリシーをアタッチできます。

キャンセル 戻る **次へ**

「新しい証明書」を選択し「次へ」ボタンをクリックします。

1-7.証明書にポリシーをアタッチ

作成した「デバイス証明書」にポリシーの関連付けを行います。

AWS IoT > 管理 > モノ > モノを作成 > 1つのモノを作成

ステップ1
モノのプロパティを指定

ステップ2 - オプション
デバイス証明書を設定

ステップ3 - オプション
証明書にポリシーをアタッチ

証明書にポリシーをアタッチ - オプション

AWS IoT ポリシーは、AWS IoT リソースへのアクセスを許可または拒否します。デバイス証明書にポリシーをアタッチすると、このデバイスからアクセスできるようになります。

ポリシー (1/7)

この証明書にアタッチするポリシーを 10 個まで選択します。

キャンセル **ポリシーを作成**

検索: kc X 1件の一致

- 名前
- KC4-policy

キャンセル 戻る **モノを作成**

先程作成したポリシー「KC4-policy」を選択し「モノを作成」ボタンをクリックします。



チュートリアル⑥

1-8.証明書とキーのダウンロード

作成された証明書とキーのファイルをダウンロードします。

証明書とキーをダウンロード

AWS に接続できるように、証明書とインストールするキーファイルをデバイスにダウンロードします。

デバイス証明書
証明書は今すぐアクティブ化することも、後でアクティブ化することもできます。デバイスが AWS IoT に接続するためには、証明書がアクティブである必要があります。

デバイス証明書

4c79109beac7cb590e8fac...te.pem.crt

キーファイル
キーファイルはこの証明書に固着しており、このページを離れるとダウンロードできません。今すぐダウンロードして、安全な場所に保存してください。

パブリックキーファイル
4c79109beac7cb590e8fac...e507ecf-public.pem.key

プライベートキーファイル
4c79109beac7cb590e8fac...507ecf-private.pem.key

ルート CA 証明書
使用しているデータエンドポイントと番号サイトのタイプに対応するルート CA 証明書ファイルをダウンロードします。ルート CA 証明書は後でダウンロードすることもできます。

Amazon 信頼サービスエンドポイント
RSA 2048 ビットキー: Amazon ルート CA 1

Amazon 信頼サービスエンドポイント
ECC 256 ビットキー: Amazon ルート CA 3

ここで必要なルート CA 証明書が表示されない場合、AWS IoT では追加のルート CA 証明書がサポートされます。これらのルート CA 証明書などは、デベロッパーガイドで入手できます。 [詳細はこちら](#)

この4つをダウンロードします。

※CA3はチュートリアルでは使用しないので、ダウンロードは任意です。

ダウンロードが完了したら「完了」をクリックします。

1-9.エンドポイントの確認

機器からAWSへアクセスするためのエンドポイントを確認します。左のメニューの「設定」を選択します。

AWS IoT

設定 情報

デバイスデータエンドポイント 情報

デバイスは、アカウントのデバイスデータエンドポイントを使用して AWS に接続できます。

各モノには、このエンドポイントで使用可能な REST API があります。MQTT クライアントと AWS IoT デバイス SDK もこのエンドポイントを使用します。

エンドポイント
iot.ap-northeast-1.amazonaws.com

ドメイン設定
ドメイン設定を作成して、デバイスの AWS IoT Core への移行、アプリケーションインフラストラクチャの AWS IoT Core への移行、ブランドアイデンティティの維持などのタスクを簡便化できます。

アクション ▼

名前	ドメイン名	ステータス	サービスタイプ	更新日
		ドメイン設定がありません。		
		ドメイン設定がありません。		

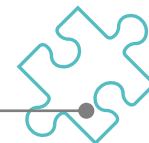
ログ 情報

CloudWatch Logs に詳細な情報を記録するために、AWS IoT ログ記録を管理できます。

デバイスからのメッセージがメッセージブローカーとルールエンジンを通過すると、AWS IoT はトラブルシューティングに役立つロギングイベントをログに記録します。

設定

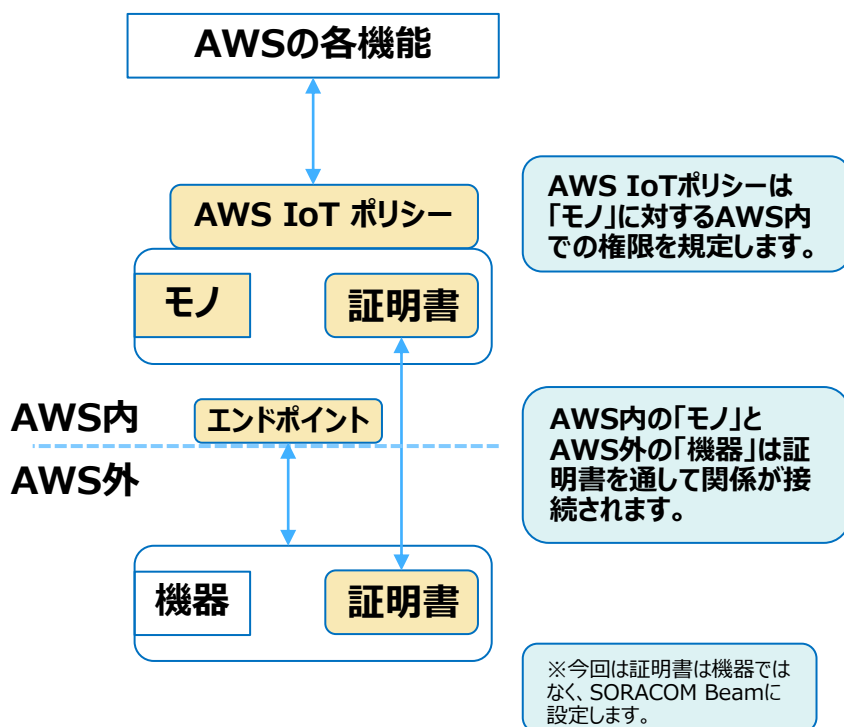
エンドポイントが表示されたら、このエンドポイントをコピーしておきます。

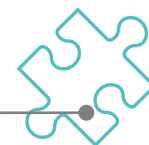


チュートリアル⑥

1-10.モノ,証明書,ポリシーの関係

作成した「モノ」、「証明書」、「ポリシー」の関係は以下の様になります。





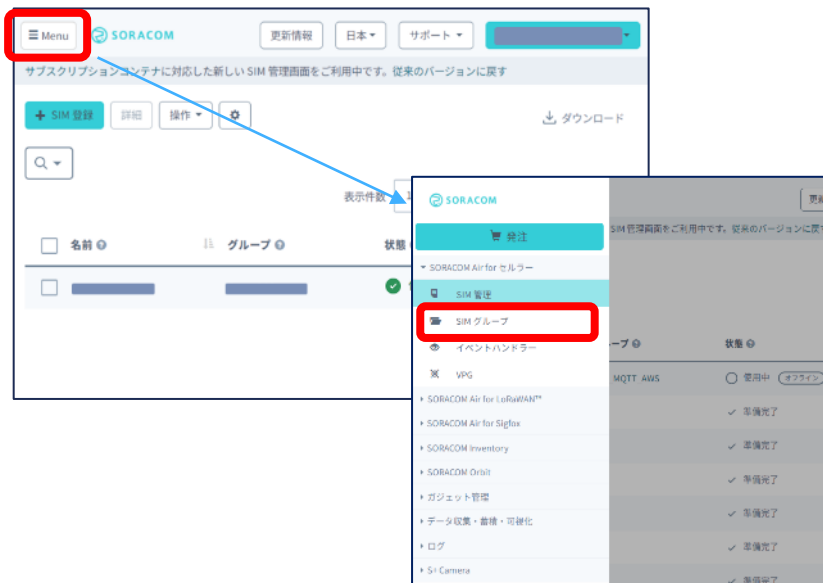
チュートリアル⑦

2.SORACOMでの転送準備

MQTTをMQTTSに変換してAWSに送るためのSORACOMの設定を行います。

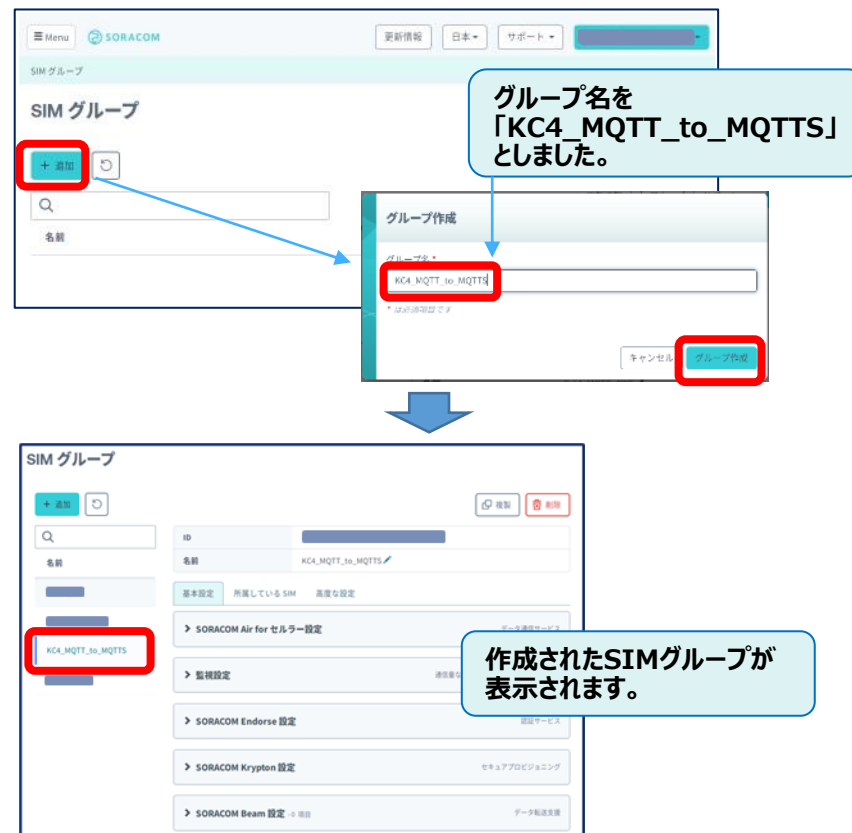
2-1.グループの作成①

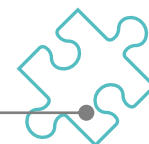
SORACOMコンソールにログインをし、左上の「Menu」を選択し、その中の「SIMグループ」を選択します。



2-2.グループ作成②

左にある「+追加」ボタンをクリックします。





チュートリアル⑧

2-3.SORACOM Beam設定①

「基本設定」の中の「SORACOM Beam設定」をクリックして、展開します。

SIM グループ

基本設定 所属している SIM 高度な設定

SORACOM Air for セルラー設定 データ通信サービス

監視設定 通信量などのイベントでアクションを実行

SORACOM Endorse 設定 認証サービス

SORACOM Krypton 設定 セキュアアプリダウンロード

SORACOM Beam 設定

IoT デバイスにかかる番号化等の高負荷処理や接続先の設定を、クラウドにオフロードもっと詳しく

名前	エントリポイント	状態	転送先	サマリ
データがありません				

HTTP エントリポイント
Web サイト エントリポイント

MQTT エントリポイント

TCP → TCP/ICMP エントリポイント

TCP → HTTP/HTTPS エントリポイント

UDP → HTTP/HTTPS エントリポイント

SMS → HTTP/HTTPS エントリポイント

2-4.SORACOM Beam設定②

「SORACOM Beam - MQTT設定」を以下のように設定します。

SORACOM Beam - MQTT 設定

設定名 ENABLED

エントリポイント

プロトコル	MQTT
ホスト名	beam.soracom.io
ポート番号	1883

転送先

種類	Standard MQTT broker
プロトコル	MQTTS
ホスト名 *	a[redacted].s.iot.ap-northeast-1.amazonaws.com
ポート番号	8883



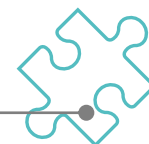
チュートリアル⑨

2-5.SORACOM Beam設定③

「SORACOM Beam - MQTT設定」の中段の「証明書」をONにして、その右にある「+」をクリックします。

ダウンロードした証明書ファイルを準備します。
※ファイル名の文字列に注目します。





チュートリアル⑩

2-6.SORACOM Beam設定④

「SORACOM Beam - MQTT設定」の下段のIMSI付与を「ON」にして「保存」をクリックします。

先程入力した認証情報IDが反映されています。

IMSI付与を「ON」にします。

「保存」を押して設定を完了します。

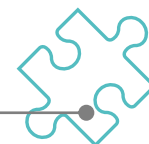
2-7.SORACOM Beam設定⑤

「SORACOM Beam 設定」に先程の設定が反映されている事を確認します。

名前	エンドリポイント	状態	転送先	サマリ
MQTTS to AWS	mqtt://beam.soracom.io:1883	有効	mqtt://a1[redacted].iot.ap-northeast-1.amazonaws.com:8883	IMSIをトピックに付与

「SORACOM」ロゴをクリックしてコンソール画面に戻ります。

グループ設定が完了したらコンソール画面に戻ります。



チュートリアル⑪

2-8.SIMへのSIMグループの適用

機器に挿入した該当のSIMにSIMグループを適用します。

このスクリーンショットは、SIM管理画面の「SIM 登録」タブに移動し、特定のSIMを選択して「詳細」ボタンをクリックする手順を示しています。赤い枠で「詳細」ボタンと「KC4-C-100A」の行が強調されています。

該当のSIMを選択して、「詳細」をクリックします。



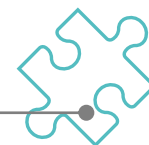
このスクリーンショットは、SIMの詳細設定画面で、既存のSIMグループ「KC4_MQTT_to_MQTTs」を選択する手順を示しています。赤い枠でグループのドロップダウンメニューが強調されています。

先程作成したグループを選択します。

該当のSIMにグループが設定されたことを確認します。

このスクリーンショットは、SIM管理画面の「SIM 登録」タブに移動し、更新されたSIMリストを確認する手順を示しています。赤い枠で「KC4_MQTT_to_MQTTs」のグループと「閉じる」ボタンが強調されています。

該当のSIMにグループが設定されたことを確認します。



チュートリアル⑫

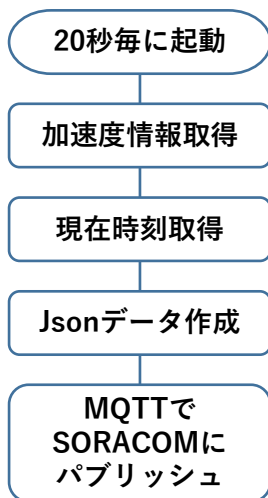
3.レシピブロックの構成

ここから機器側のレシピを作成していきます。

レシピの新規作成やブロックの配置は、導入編やHTTPのチュートリアルをご参照ください。

3-1.レシピの構成

本レシピは以下の流れで構成します。



3-2.加速度/現在時刻情報の取得

加速度ブロックを使用して、XYZの数値を取得

加速度データ取得

タイムアウト(1~2147483) 60 秒

取得完了

- 数値変数 acc_x に加速度データ X軸の加速度 をセット
- 数値変数 acc_y に加速度データ Y軸の加速度 をセット
- 数値変数 acc_z に加速度データ Z軸の加速度 をセット

取得失敗

現在時刻ブロックを使用して時刻を取得

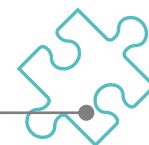
現在時刻を取得

タイムアウト(1~2147483) 60 秒

実行処理

- 数値変数 time にUNIX時間をセット

失敗処理



チュートリアル⑬

3-3. 取得情報のJson文字列化

マップ変数で構成したデータをJson文字列に変換します。

マップ変数 `get_info` を初期化
容量(1-32) 4 KByte
データ(key, value)

- key名: `"ACC_X"` value: 数値変数 `acc_x`
- key名: `"ACC_Y"` value: 数値変数 `acc_y`
- key名: `"ACC_Z"` value: 数値変数 `acc_z`
- key名: `"TIME"` value: 数値変数 `time`

文字変数 `map_to_json` を最大 1280byte のサイズで定義
文字変数 `map_to_json` にマップ変数 `get_info` をJSON形式の文字列でセット

マップ変数にて、keyとvalueのデータ構成を行い、その構成を文字列に置き換える処理を行います。

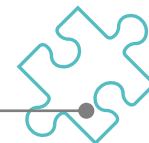
3-4. MQTT送信

MQTTパケットをSORACOM Beam宛に送信を行います。

MQTTブローカーとクライアントID `"ClientID1"`
サーバアドレス `beam.soracom.io`
ポート番号(0~65535) 1883
keepalive(0~65535) 100 秒
Clean Session True
QoS 0
ユーザー名
パスワード
タイムアウト(1~2147483) 7 秒
ブローカー接続成功

トピックにメッセージをパブリッシュ
トピック `"topic"`
メッセージ 文字変数 `map_to_json`
データ送信失敗時 送信データを破棄
タイムアウト(1~2147483) 7 秒
成功 ディスプレイへ文字を表示
PUB_SUCCESS
失敗 ディスプレイへ文字を表示
PUB_FAIL
MQTT接続を切断
ブローカー接続失敗 ディスプレイへ文字を表示
BROKER_FAIL
last will topic
last will message

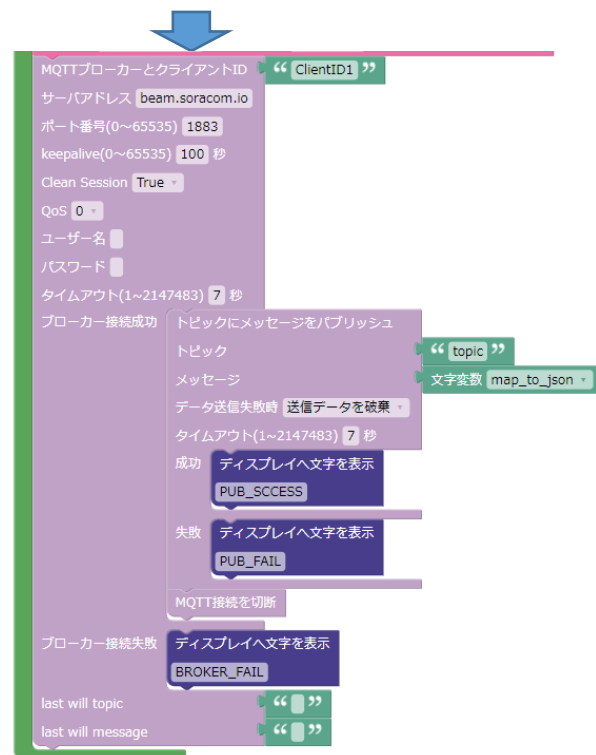
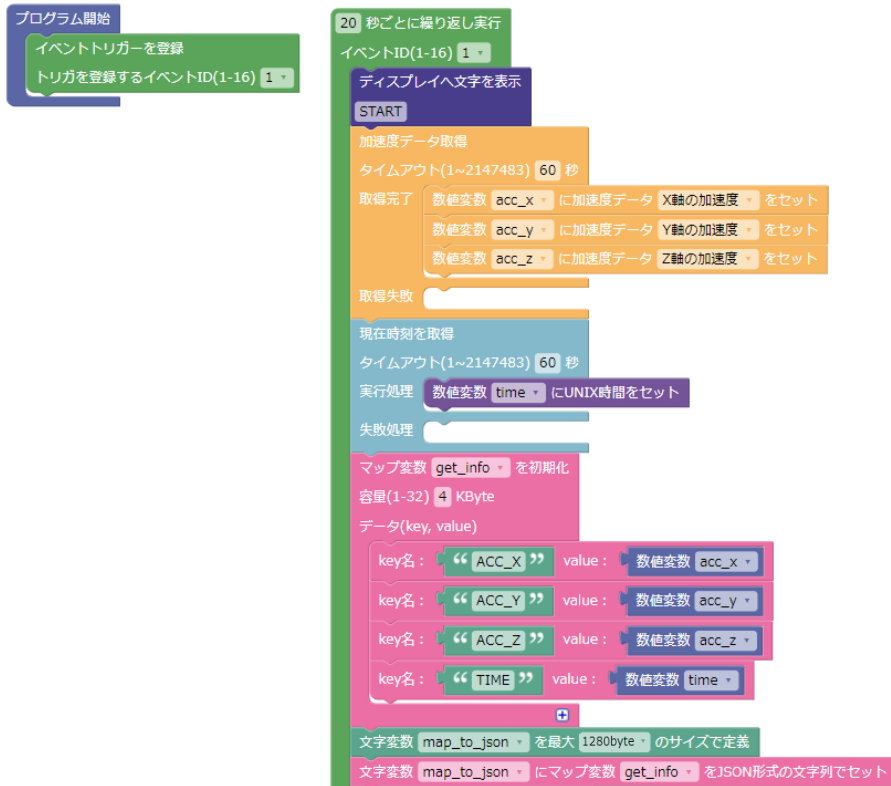
SORACOM Beamのアドレスを指定。
ポートは1883
topic/というトピックの下にJsonメッセージを添付



チュートリアル⑭

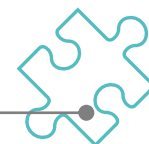
3-5. レシピの全体

構成したレシピ全体は以下の通りになります。



レシピを構成したら、機器への書き込みを行い動作確認を行います。

機器への書き込みは「導入編」等ご参照お願い致します。



チュートリアル⑮

4.MQTTのAWS到達確認

SORACOM Beam経由でAWSにMQTTパケットが到達している事を確認します。

AWSのコンソールにて以下を選択。

トピックのフィルター 情報
トピックフィルタは、サブスクライブするトピックのフィルターを入力

トピックをサブスクライブする

トピックのフィルター 情報
トピックフィルタは、サブスクライブするトピックのフィルターを入力

追加設定

サブスクライブ

サブスクライブ

トピックのサブスク립ションがありません。

受信メッセージを表示する

「MQTTテストクライアント」を選択

「トピックをサブスクライブする」画面が表示されるのを確認

「topic/#」と入力して「サブスクライブ」ボタンをクリック

データが到達したタイミングでJsonデータが表示されます。

サブスク립ション topic/#

topic/#

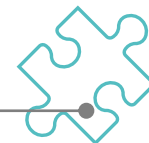
topic/#

```
{
  "ACC_X": -13.42,
  "ACC_Y": -12.2,
  "ACC_Z": 1013.332031,
  "TIME": 1648025526
}
```

topic/#

```
{
  "ACC_X": -11.224,
  "ACC_Y": -14.64,
  "ACC_Z": 1016.992004,
  "TIME": 1648025506
}
```

topic/xxxxxxxxxxxx のxxの数字の部分は、SORACOM Beamで付与されたIMSIとなります。



チュートリアル①⑥

5. MQTTブロックチュートリアルの終了

以上でMQTTのチュートリアルを終了します。

お疲れさまでした。

今回は2つのクラウドサービスを利用したため、少し複雑な設定が必要でしたが、MQTTブロックへの設定項目は少なく、シンプルに使えたと思います。

一度、クラウド側にデータが届くことが確認出来たあとは、他のセンサーなどのデータを追加することも簡単ですので、他の機能と共に色々なレシピを試してみてください。



THE NEW VALUE FRONTIER



京セラ株式会社

© 2022 KYOCERA Corporation

- ※ 本資料は2022年6月現在のものです。
- ※ LTEは、ETSIの商標です。
- ※ ソラコム、SORACOM、又はソラコムの商品・サービス名称等は、株式会社ソラコムまたはその関連会社の商標または登録商標です。
- ※ Amazon Web Services、アマゾン ウェブ サービス、AWS、AWS IoT Coreは、米国その他の諸国における、Amazon.com, Inc. またはその関連会社の商標です。