MGCP ユーザーマニュアル

MGCP User Manual Rev A.1.41
MGCP version: v1.4b01
2023/2/26

オールブルーシステム (All Blue System)
ウェブページ: www.allbluesystem.com
コンタクト:contact@allbluesystem.com

1　はじめに

このマニュアルでは MGCP ファームウエアを搭載した Espressif Systems[1]社製 ESP32 プロセッサ用のリモートコントロールデバイス(以降 “MGCPデバイス” と略す) の機能や使い方について詳しい情報を提供します。

2　表記について

2.1　著作権および登録商標

2.2　連絡先

オールブルーシステム (All Blue System)
ウェブページ http://www.allbluesystem.com
メール contact@allbluesystem.com

3　イントロダクション・機能概要

MGCP デバイスを使用することで Wi-Fi ネットワーク内に設置する、センサーデバイスや I/O 装置、アクチュエータ、表示器等を簡単に作成することができます。リモートデバイスを構築するときのファームウエア作成や通信ネットワーク構築などの手間をかけずに、簡単に複数のデバイスを使用したセンサーネットワークを設置できます。

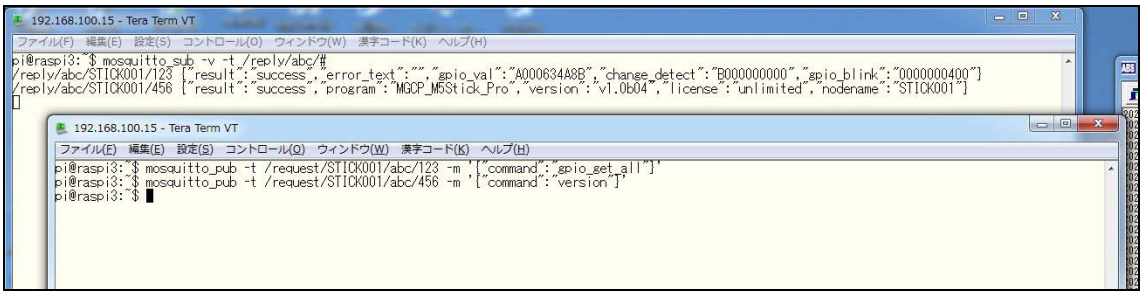
MGCP デバイスは、汎用のESP32 プロセッサを搭載した CPU ボードや無線モジュールに専用のファームウエアを書き込むことで簡単に作成できます。MGCP デバイスはリモート制御用に必要となるコンフィギュレーション保存機能やリモートコマンド制御、ペリフェラル制御機能、ログ出力などの機能が予め組み込まれていますので、直ぐにユーザーの環境に設置して様々なセンサーやアクチュエータを使用したシステムを構築できます。

MGCP デバイスには以下のような特徴・機能があります。

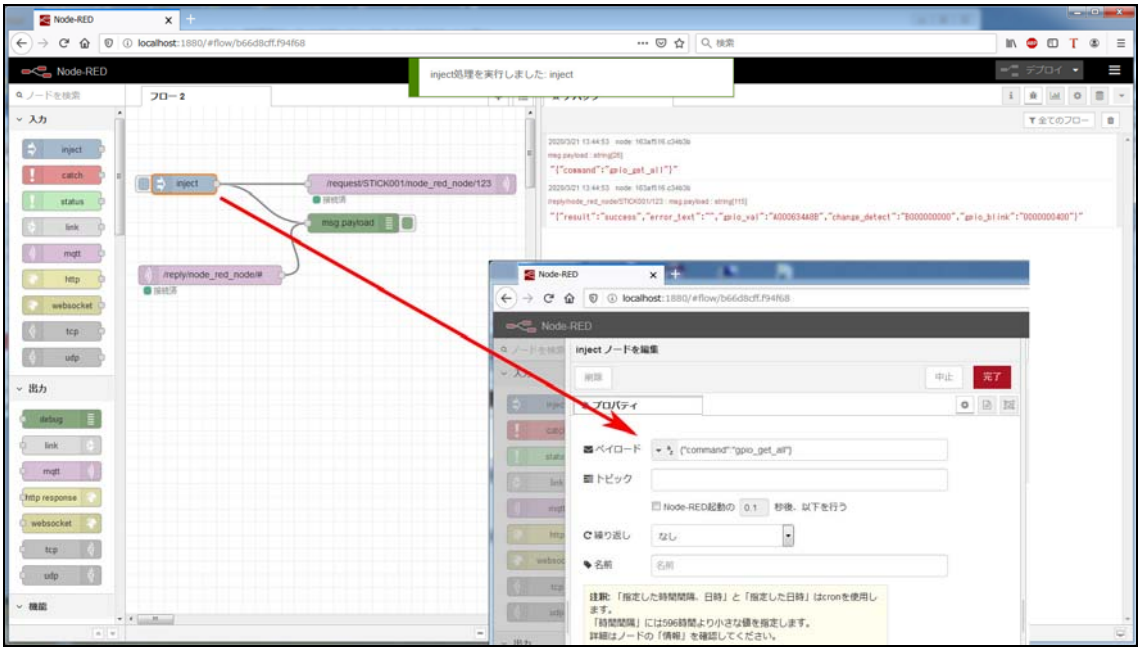
■ MGCPデバイスに対するコマンド送信やリプライ受信、イベントデータ受信は全て MQTT ブローカ経由で行います。このため、MGCP デバイスは MQTTプロトコルをサポートする様々なシステムから簡単にリモート操作することができます。コマンドやリプライ、イベントデータのフォーマットはJSON フォーマットになっていますので様々なシステ

[1] https://www.espressif.com/

ムで簡単に扱うことができます。また、パケット送受信には標準的な MQTTプロトコルを使用しますので、既存のシステムに簡単に接続してリモートセンサーネットワークを構築できます。



(mosquitto[2] MQTT クライアントを使用して、リモートMGCP デバイス操作している様子)

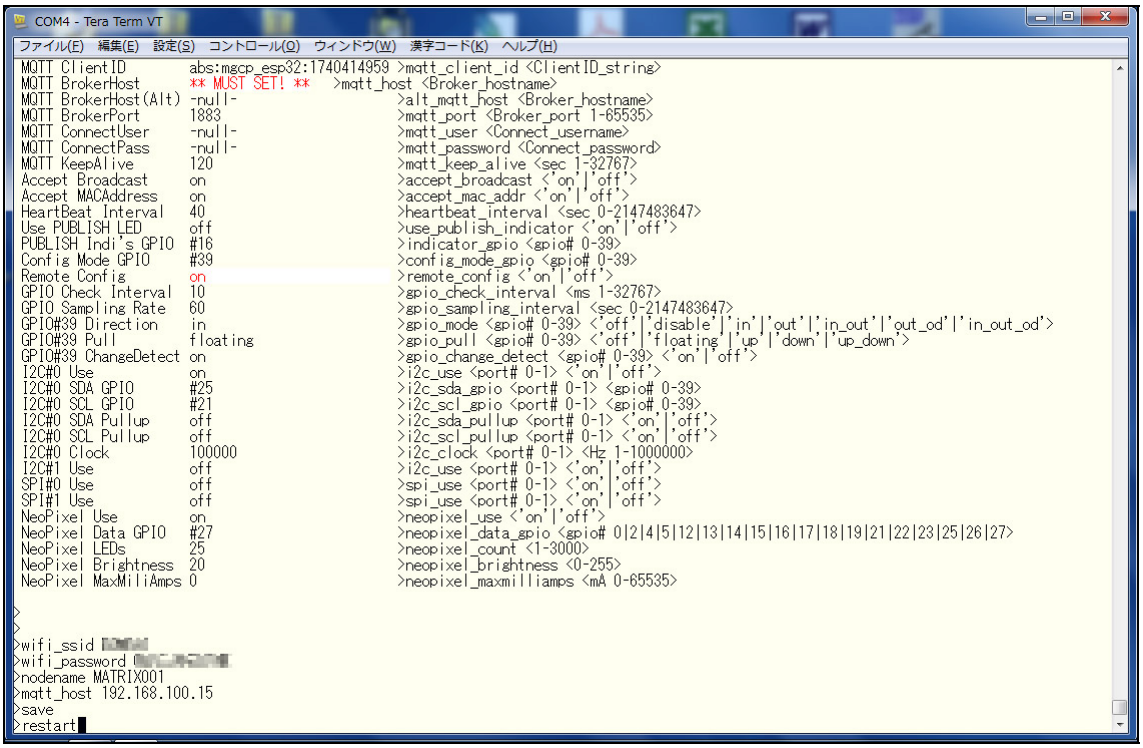


(Node-Red[3] からリモートMGCP デバイスを操作している様子)

■ コンフィギュレーション・コンソールモードによる各種設定機能

MGCPデバイスは通常運用時は Wi-Fi 接続経由でリモートから操作可能ですが、Wi-Fi 接続設定や SPIポート、I2Cバス設定等のために、シリアルポート経由でコンフィギュレーションの確認や設定を行うコンフィギュレーション・コンソール・モードも用意しています。

[2] https://mosquitto.org/
[3] https://nodered.org/



（コンソールモードでデバイスの初期設定を行っている様子）

■ 汎用の ESP32デバイス用と、M5Stack[4]社製の M5Stack,M5Stick-C,Atom-Lite,Atom-Matrix専用ファームウエアを提供

MGCP プログラムは ESP32プロセッサを使用しているマイコンボードや無線モジュールで動作させることができます。GPIOポートのモードやI2C, SPI のバス設定等も自由に設定できますので、使用するデバイスの仕様に合わせることができます。

M5Stack や M5Stick-C等の LCDを搭載したモジュール向けには、内蔵 LCD ディスプレイ用の機能が追加された、専用の MGCPファームウエアを提供しています。内蔵ボタンや NeoPixel[5]互換 LED 表示用に予めコンフィギュレーションのデフォルト値が設定されていますので、これらのペリフェラルを直ぐに使用することができます。

■ リモートから GPIO ポートを操作することができます。GPIOポート値のリードとライト以外も、GPIOモードの変更やプルアップ・ダウン、変化検出等の設定をリモートから制御できます。また、GPIOのブリンク(On-Off繰り返し)状態設定や、パルス信号出力をリモートから指定できます。

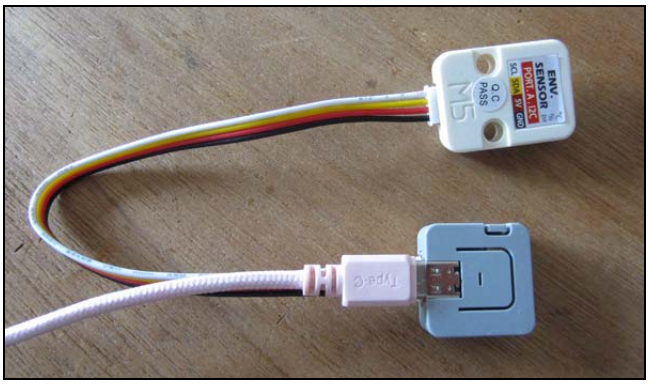
M5Stack,M5Stick-C,Atom-Lite,Atom-Matrix 用の MGCP デバイスでは、内蔵ボタンのイベント(押し込んだタイミングと放したタイミングの両方)を検出してイベント送信するための GPIOポートが予め設定されています。また、デバイス内蔵の LED 等も予め GPIOポートが設定済みなので、直ぐにリモートから制御することができます。
また、負論理で接続するI/O用に、GPIO の初期値を設定することもできます。

[4] https://m5stack.com/
[5] "NeoPixel" is Adafruit's brand, https://www.adafruit.com/, https://www.adafruit.com/category/168

```
GPIO#10 Direction    out                >gpio_mode <gpio# 0-39> <'off'|'disable'|'in'|'out'|'in_out'|'out_od'|'in_out_od'>
GPIO#10 Initial      1                  >gpio_init <gpio# 0-39> <'off'|'0'|'1'>
GPIO#36 Direction    in                 >gpio_mode <gpio# 0-39> <'off'|'disable'|'in'|'out'|'in_out'|'out_od'|'in_out_od'>
GPIO#36 Pull         up                 >gpio_pull <gpio# 0-39> <'off'|'floating'|'up'|'down'|'up_down'>
GPIO#36 ChangeDetect on                 >gpio_change_detect <gpio# 0-39> <'on'|'off'>
GPIO#37 Direction    in                 >gpio_mode <gpio# 0-39> <'off'|'disable'|'in'|'out'|'in_out'|'out_od'|'in_out_od'>
GPIO#37 Pull         floating           >gpio_pull <gpio# 0-39> <'off'|'floating'|'up'|'down'|'up_down'>
GPIO#37 ChangeDetect on                 >gpio_change_detect <gpio# 0-39> <'on'|'off'>
GPIO#39 Direction    in                 >gpio_mode <gpio# 0-39> <'off'|'disable'|'in'|'out'|'in_out'|'out_od'|'in_out_od'>
GPIO#39 Pull         floating           >gpio_pull <gpio# 0-39> <'off'|'floating'|'up'|'down'|'up_down'>
GPIO#39 ChangeDetect on                 >gpio_change_detect <gpio# 0-39> <'on'|'off'>
```

(M5Stick-C 用MGCP デバイスの GPIO初期設定例。内蔵LEDとボタンの他、PIR-HAT用の追加設定をした様子)

■ SPI, I2C バスに接続したデバイスをリモートから操作可能で、任意データの送受信が可能。



(デバイス内蔵のGroveポート経由でI2Cセンサーを接続した様子)

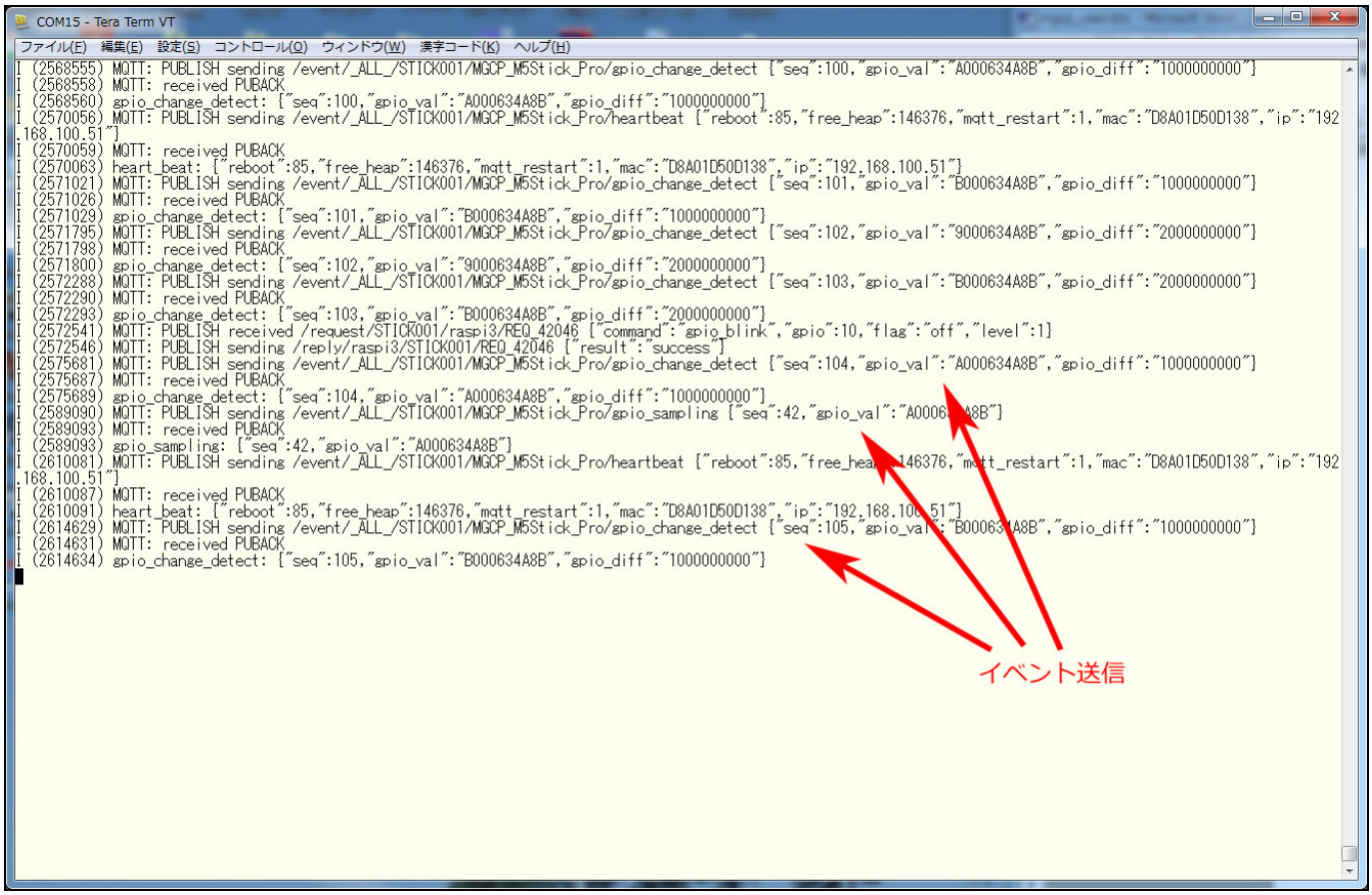
リモートから I2C バスに接続したデバイスの書き込みや読み込み、SPI ポートへのデータ送受信が可能です。ブロック毎に書き込み動作を行うコマンドも用意していて、ブロック書き込み間にディレイを入れるなど短いタイミング制御が必要なデバイスにも対応しています。

I2C, SPI で使用する GPIOポートや通信条件もコンフィギュレーションコンソールまたは、リモートから実行するMGCPコマンドで自由に設定することができます。

```
I2C#0 Use            on                 >i2c_use <port# 0-1> <'on'|'off'>
I2C#0 SDA GPIO       #21                >i2c_sda_gpio <port# 0-1> <gpio# 0-39>
I2C#0 SCL GPIO       #22                >i2c_scl_gpio <port# 0-1> <gpio# 0-39>
I2C#0 SDA Pullup     off                >i2c_sda_pullup <port# 0-1> <'on'|'off'>
I2C#0 SCL Pullup     off                >i2c_scl_pullup <port# 0-1> <'on'|'off'>
I2C#0 Clock          100000             >i2c_clock <port# 0-1> <Hz 1-1000000>
I2C#1 Use            on                 >i2c_use <port# 0-1> <'on'|'off'>
I2C#1 SDA GPIO       #0                 >i2c_sda_gpio <port# 0-1> <gpio# 0-39>
I2C#1 SCL GPIO       #26                >i2c_scl_gpio <port# 0-1> <gpio# 0-39>
I2C#1 SDA Pullup     off                >i2c_sda_pullup <port# 0-1> <'on'|'off'>
I2C#1 SCL Pullup     off                >i2c_scl_pullup <port# 0-1> <'on'|'off'>
I2C#1 Clock          100000             >i2c_clock <port# 0-1> <Hz 1-1000000>
```

(M5Stick-C 用MGCP デバイスの I2Cバス初期設定例)

■ GPIOポート値の変化や定期的なGPIOポート値の送信(サンプリングタイマー)、デバイス起動・リスタート、ハートビート(定期送信)など予め決められた条件になったときにイベントデータを送信します。

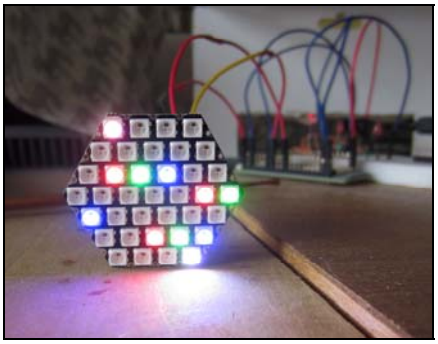


(PIRセンサーやボタン入力によるイベント送信、GPIOサンプリング、ハートビート・イベントを送信している様子)

■ NeoPixel 互換 LED の点灯や点滅をリモートから操作できます。

NeoPixel 互換 LED を GPIO ポートに接続して、フルカラーLEDテープ等をリモートからコントロールすることができます。Atom-Lite、Atom-Matrix 内蔵の NeoPixel LED は予めコンフィギュレーション済みなので、インストール後にすぐに使用できます。

個々の LED を任意の色に設定したりブリンク状態にすることができます。LED全体の明るさや電流制限値の設定、複数のLED の色を一度の MGCPコマンドで設定することも可能です。



(ESP32 CPU ボードに NeoPixel LEDモジュールを接続した様子)

(Atom-Matrix の 内蔵LED をリモートから操作している様子)

■ PWM波形出力や SERVO デバイスをリモートから操作できます。

PWM 波形の出力を GPIO ポートから出力できます。最大４チャンネルまで同時に PWM波形の周波数やデューティ値を任意の値に設定できます。また、PWM 信号を簡易的なブザー出力に使用するときに便利な、間欠出力機能も提供しています。

サーボデバイス操作用に SERVO信号に適した PWM 設定用のコマンドも使用できます。サーボ位置や周波数を任意の値にリモートから簡単に操作できます。

(MGCPデバイスに接続可能なサーボデバイス例)

■ LCD 表示器を内蔵したデバイスでは、文字や図形をリモートから描画できます。

M5Stack, M5Stick-C 内蔵の LCD に図形や文字を描画できます。リモートから１つの MGCP コマンドで複数の図形を一度に描画したり、複数のコマンドを実行して様々な情報を表示することができます。

LCD に任意のビットマップデータを指定してパターンを表示することもできます。クライアント側でビットマップ・フォントデータを使用した、文字コードからビットマップデータへの変換機能を用意することで、様々な日本語フォントを使用した表現も可能です。

（リモートから図形と文字を描画している様子）

■ インターネットや LAN を跨ぐ構成でリモートアクセス可能。

MGCP デバイスは、コンフィギュレーション設定で指定した MQTT ブローカと接続することでリモートコマンドの受信やリプライ送信、イベント送信を行っています。このため、MGCP デバイス側からアクセス可能な LAN またはインターネット上の MQTT ブローカを指定することで、LAN や インターネット環境からリモート操作が可能です。

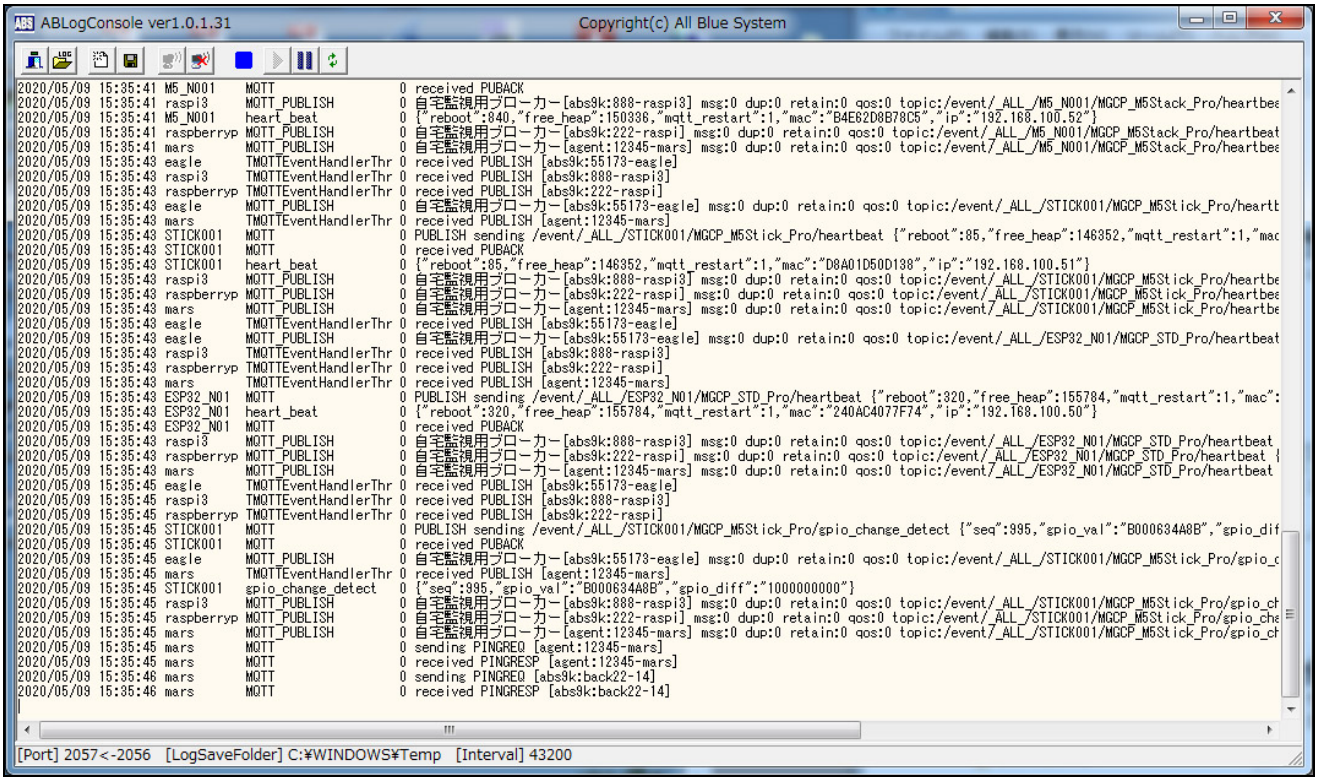
プライベート LAN 内に MGCP デバイスを設置して、それらをインターネットからアクセスする場合でも、デフォルトのゲートウェイやルータ設定のみで、特別なポート解放設定やフォワード設定、グローバルアドレスの付与などは不要です。

■ インターネットとは切断された環境の、完全に閉じたローカルネットワーク内での動作も可能。

MGCP デバイス動作時にはインターネットアクセスできない環境でも問題なく動作させることができます。セキュリティ要件が厳しい企業や工場、病院施設等の、インターネットに直接アクセスできない環境でも問題なく運用することができます。

■ MGCP デバイスが出力するログメッセージを専用のログサーバーで集中管理することができます。

MGCPデバイスを設置したネットワーク内にオールブルーシステムが提供するログサーバー(ABS-9000 LogServer)を設置することで、複数の MGCP デバイスのログをリアルタイムに確認したり、ログメッセージを保存しておくことができます。ABS-9000 LogServer はオールブルーシステムのホームページからダウンロードして、フリーで使用することができます。



(ABS-9000 LogServerで複数の MGCPデバイスが出力しているログメッセージを集中管理している様子)

MGCP デバイスのログメッセージは ESP32のシリアルポートにも同時に出力しています。このため、ネットワーク障害中でも仮想COM ポート経由で MGCPデバイスのログメッセージを確認することができます。

■ MGCP デバイスの IPアドレスは、LAN 内に設置した DHCP サーバーによる自動付与で運用できます。この場合でもデバイス毎にユーザーが設定したノード名によるリモートデバイス指定が可能です。

■ MQTT ブローカとの通信に失敗した場合には、MQTT ブローカへの再接続を自動的に行います。予備の MQTTブローカへの自動切り替えを設定することもできます。Wi-Fi ネットワークが切断した場合など、デバイス側で致命的な障害を検知すると自動的にデバイスがリスタートします。デバイスの電源断によるリスタート時などの場合でもクライアント側では特別な操作は必要なく、通信可能になった時点で全ての機能が使用可能な状態になります。

■ 複数の MGCP デバイスと、デバイスを利用する複数のクライアントコンピュータを同一 MQTT ブローカに収容して、大規模なセンサーネットワークを簡単に構築・管理できます。

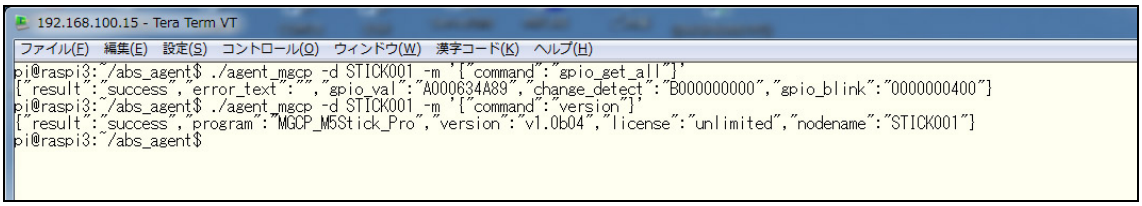
クライアント(デバイス制御側)とMGCPデバイスを互いに複数同時にアクセスするようなシステム上で、単体のクライアントや MGCP デバイスがアクセス不能になったときでも、他のクライアントやデバイス間の通信には問題が発生しないような分散処理システムを簡単に構築できます。

■ オールブルーシステム製の abs_agent で提供するライブラリ関数をコールすることで、リモートコマンド実行やイベントハンドラ処理、日本語文字列表示を簡単に実現できます。同梱のクライアントプログラム(agent_mgcp)を使

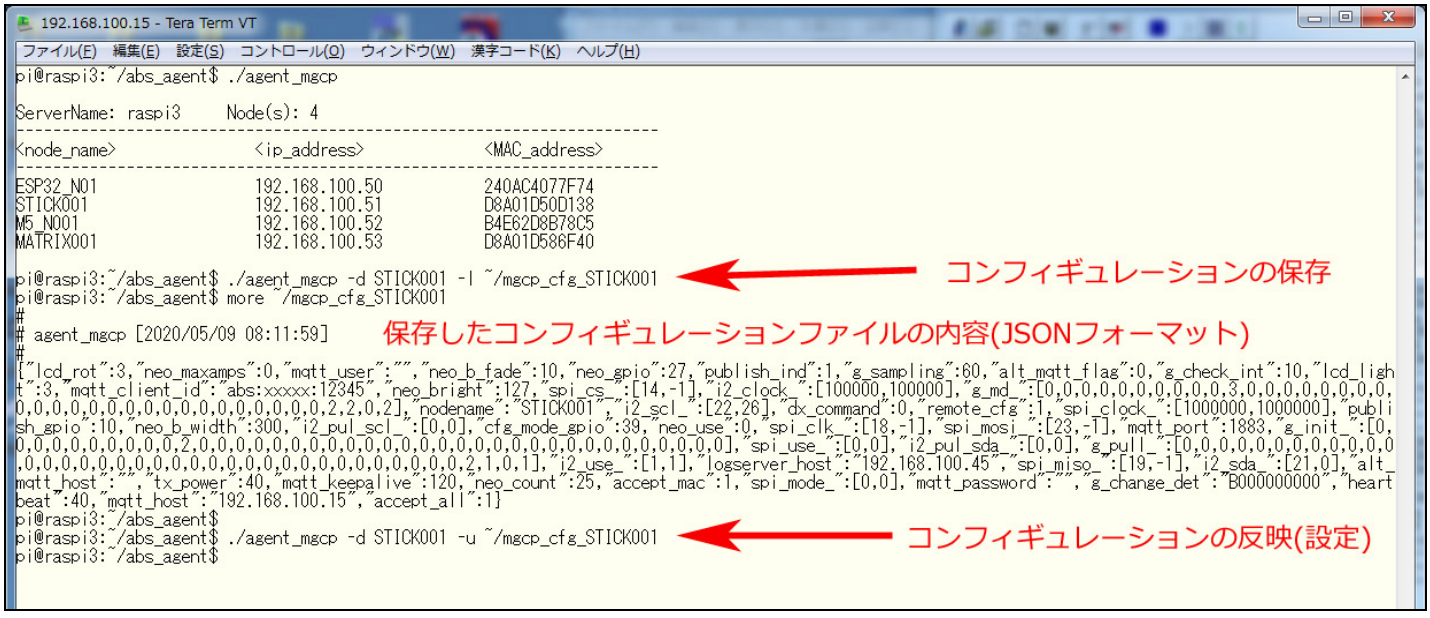
用すると、MGCPデバイスの一元管理やリモートコマンド実行、コンフィギュレーションの保存・設定がリモートから実行できます。

(M5Stick-C用 MGCP デバイスのLCDに日本語メッセージを表示している様子)

日本語フォントは abs_agent 側の共有メモリエリアに格納した任意のビットマップ・フォントを使用しています。文字コードを元にビットマップデータに変換して、リモートMGCPコマンドを使用して描画しています。日本語表示用 API は M5Stack, M5Stick-C 用の MGCP デバイスが対応しています。詳しくは "MGCPデバイス操作ライブラリ関数" の項を参照してください。



(agent_mgcp プログラムを使用してリモートMGCP デバイスのGPIO値やバージョンを取得している様子)



(リモートMGCP デバイスのコンフィギュレーションをファイルに保存したり、保存済みのコンフィギュレーション値をリモートMGCP デバイスに反映している様子)

コンフィギュレーション値をファイルに保存することで、MGCPデバイスのハードウエアを入れ替える時の作業が簡単

になります。また、デバイス設置場所で仮想シリアルポート経由でコンフィギュレーション・コンソールを利用しなくても、リモートから MGCPデバイスのコンフィギュレーション値の確認や設定変更ができます。

4 MGCPデバイス・インストール

MGCP デバイスには専用のファームウエアを書き込んで使用します。Espressif Systems[6]社製 ESP32 プロセッサを使用している CPU ボードやモジュールに MGCP ファームウエアを書き込むことができます。

M5 Stack[7]社製の M5Stack,M5Stick-C,Atom-Lite,Atom-Matrix 向けには、内蔵デバイスに直ぐにアクセスできるように初期設定済みの MGCP ファームウエアを提供しています。LCDディスプレイや NeoPixel LEDにメッセージを表示したり、搭載しているボタン入力に応答するシステムを簡単に構築できます。Grove ポートには様々なセンサーをI2Cや GPIO ポート経由で簡単に接続できます。

⚠ **ESP32プロセッサやフラッシュメモリ中のブートローダー・モニタプログラム・ユーザープログラム等の既存プログラムやデータは上書きされます**

この章で説明する手順で MGCP デバイス用のファームウエアを書き込むと、ESP32プロセッサやフラッシュメモリに書き込み済みのファームウエア(ブートローダーやモニタプログラム、ユーザープログラム)は全て上書きされます。また、不揮発領域(NVS) もMGCPデバイスから利用しますので、既存の NVSデータにはアクセスできなくなる可能性があります。もし、MGCP デバイスで使用した後にデバイス購入時のファームウエアに戻したい場合には、デバイス購入元のドキュメントやサポート情報を参照してファームウエアやアプリケーションデータを書き込む必要があります。オールブルーシステムではこれらの復元手順や方法については関知できません。復元が必要な場合には、予めこれらの情報を入手してから MGCP デバイス用のファームエアを書き込む事をお勧めします。

4.1 MGCPデバイス動作環境

MGCPプログラムは下記の環境で動作します。

項目	必要な機材やデバイス
CPU ボード(無線モジュール)	* Espressif Systems社製 ESP32プロセッサを搭載した任意の CPU ボード (シリアルポート経由でファームウエア書き換えが可能な事) * M5Stack製の以下の製品用には、専用の MGCPファームウエアを提供 "M5Stack BASIC/Grey" その他 M5Stack互換製品 "M5Stick-C" "Atom-Lite" "Atom-Matrix"
シリアルポート	ESP32 プロセッサのシリアルポートを PC 等に接続するためのインターフェイ

[6] https://www.espressif.com/
[7] https://m5stack.com/

	スが必要です。 シリアルポートは、セットアップ時のファームウエア書き込みと、初期設定用のコンフィギュレーションコンソールで使用します。 運用中は、デバイスのログメッセージがシリアルポートに出力されます。
ネットーワーク	Wi-Fi アクセス可能な無線アクセスポイント。 無線アクセスポイントからMQTT ブローカやログサーバーへ接続するために必要な、ルーティングまたはブリッジ機能を有する事。
DHCP サーバー	MGCP デバイスはLAN 内に設置した DHCP サーバーより提供された任意のローカル IP アドレスを使用して起動します。デフォルトゲートウェイや DNS サーバー情報も DHCP サーバーから自動取得します。 通常は Wi-Fi アクセスポイントやルーター装置に内蔵されています。
MQTT ブローカ	MQTT プロトコル ver3.1(3.1.1) を実装した MQTT ブローカが必要です。 Open Source で開発されている Mosquitto[8] 等が動作しているコンピュータをLAN内に設置してください。MQTT ブローカには 固定 IPアドレスでアクセスできるようにしてください。 Internet、WAN 上に MQTT ブローカが設置されている場合には、これを利用することもできます。この場合にはドメイン・ホスト名やグローバル IP アドレスで LAN 内からアクセス可能なこと。
ログサーバー(オプション)	ABS-9000 LogServer プログラムを設置した Windows コンピュータ LAN/WAN 内で固定 IPアドレスでアクセス可能な事。 MGCP デバイスで設定したログサーバーが運用中に停止した場合でも、MGCPデバイスは正常に動作します。

4.2 MGCP で使用するネットワークポート

MGCP では MQTT ブローカーとの通信やログ出力用にネットワークポートを使用します。詳しい仕様は以下になります。

MGCPプログラムが使用するプロトコルとポート番号		
プログラム・機能	**プロトコル**	**ポート番号**
MQTT ブローカとの通信 コマンドリクエスト受信とリプライ、イベント情報送信	TCP (MQTT)	1883 (変更可能)
ログサーバーへのメッセージ送信	UDP	2056

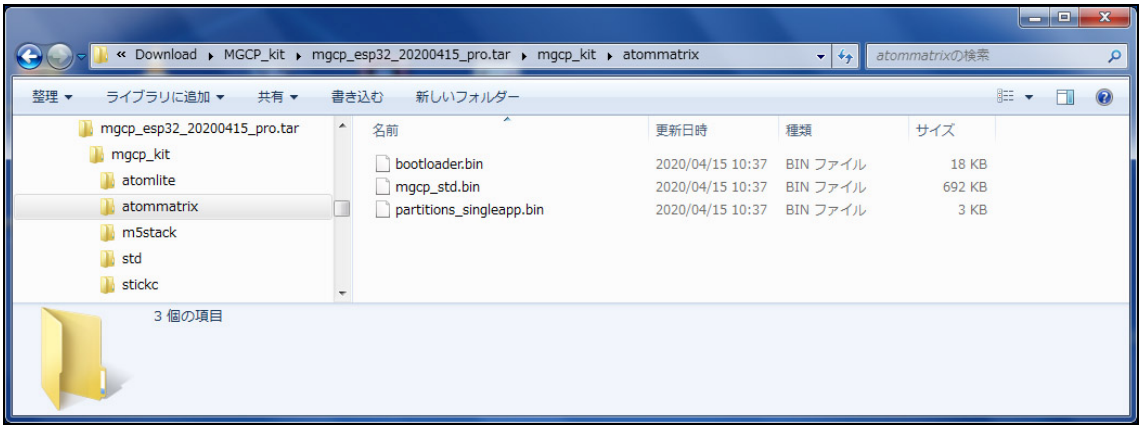
4.3 MGCPファームウエアダウンロード

MGCP デバイスのインストール手順では、ファームウエア書き込み作業用に Windows コンピュータを使用します。予め MGCP ファームウエアを作業用 Windowsコンピュータにダウンロードしておきます。

[8] https://mosquitto.org/

オールブルーシステムのホームページのダウンロード画面から、MGCP ファームエアをダウンロードします。
ファームウエアは "デモ版"と "正規版"の２つがありますので、ライセンスを入手されている方は "正規版"を選択してください。"デモ版" は何方でもダウンロード可能です。

ダウンロードしたファームウエアは (tar + gzip)形式でアーカイブされています。Windows 上で動作するフリーソフト(7Zip[9], +Lhaca[10])等を利用すると簡単にアーカイブ・ファイルを展開できます。

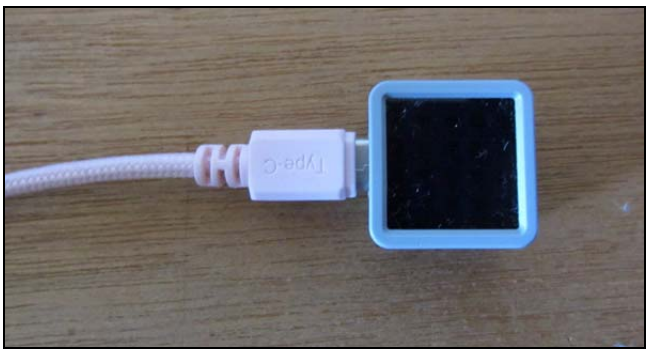


(ダウンロードしたファームウエアを展開した様子)

展開したファームウエアはデバイスタイプ毎にディレクトリが分かれていますので、使用するデバイスに合わせて選択します。
現在、M5Stack, M5Stick-C, Atom-Lite, Atom-Matrix 用に専用のファームエアを提供しています。それ以外の ESP32 プロセッサを使用したデバイスには汎用の "std" タイプを選択してください。

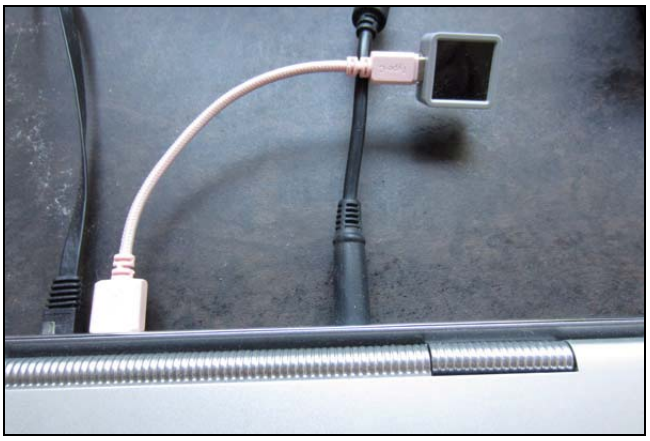
4.4 MGCPファームウエア書き込みと初期設定

このマニュアルではインストール例として、M5Stack社の Atom-Matrix に MGCP ファームウエアを書き込みます。その他の 汎用のESP32 CPUボードや M5Stack, M5Stick-Cでも選択するファームウエアが違うだけでセットアップ方法は全く同じです。


(ESP32プロセッサを搭載した M5Stack社製の Atom-Matrix)

[9] https://sevenzip.osdn.jp/
[10] https://forest.watch.impress.co.jp/library/software/pluslhaca/ (ページ中に公式ページへのリンクあり)

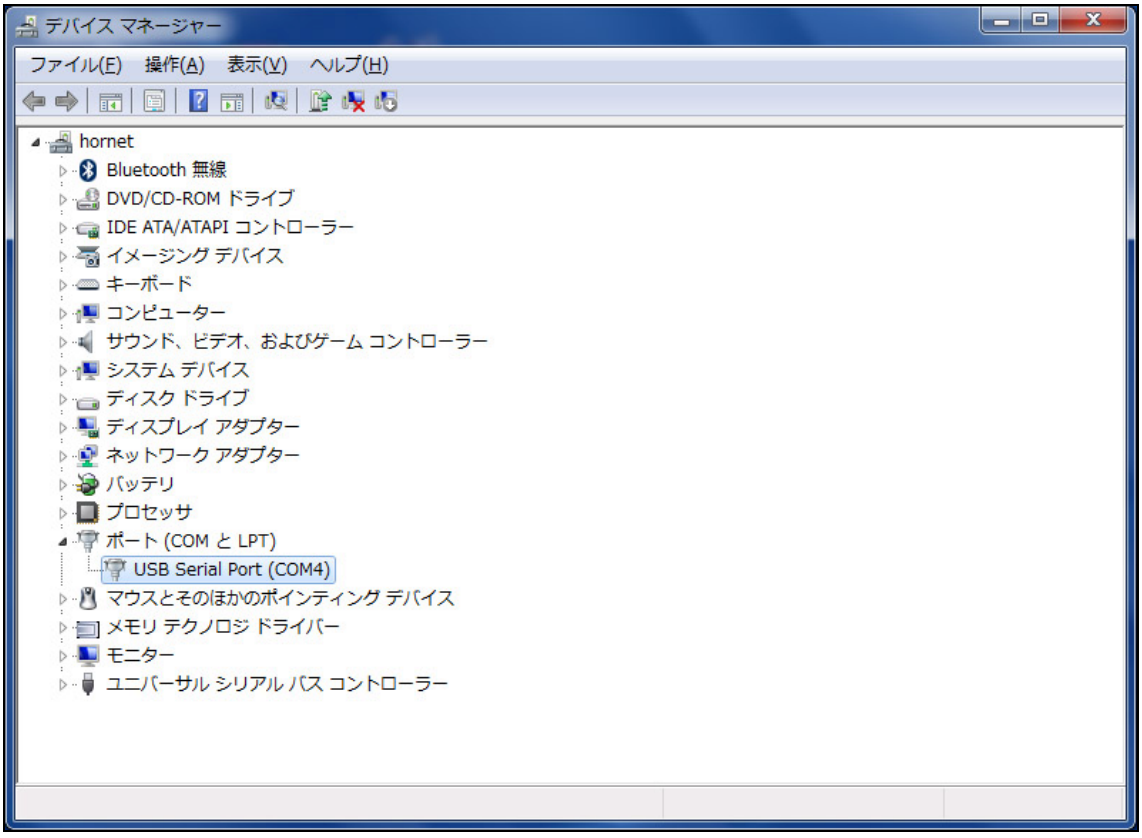
USB ケーブルを使用して Atom-Matrix と書き込み作業用の Windows コンピュータを接続します。



(USBケーブルで Windows コンピュータに Atom-Matrix を接続した様子)

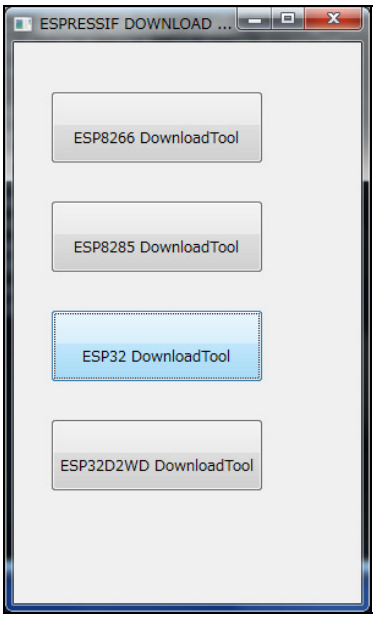
USBポートに接続すると、Windows 上では仮想 COM ポートデバイスが自動作成されて "COMxx"(xx は数字)としてデバイスマネージャに表示されます。

インストール対象のESP32デバイスに搭載している USB-シリアル変換IC の種類によっては、Windows コンピュータに変換IC用のシリアルポート・ドライバのインストールが必要になる場合があります。詳しくは ESP32 CPU ボードメーカのドキュメントを参照してください。



(Windows 上で Atom-Matrix に接続された仮想COM ポートが表示されている。この例では "COM4")

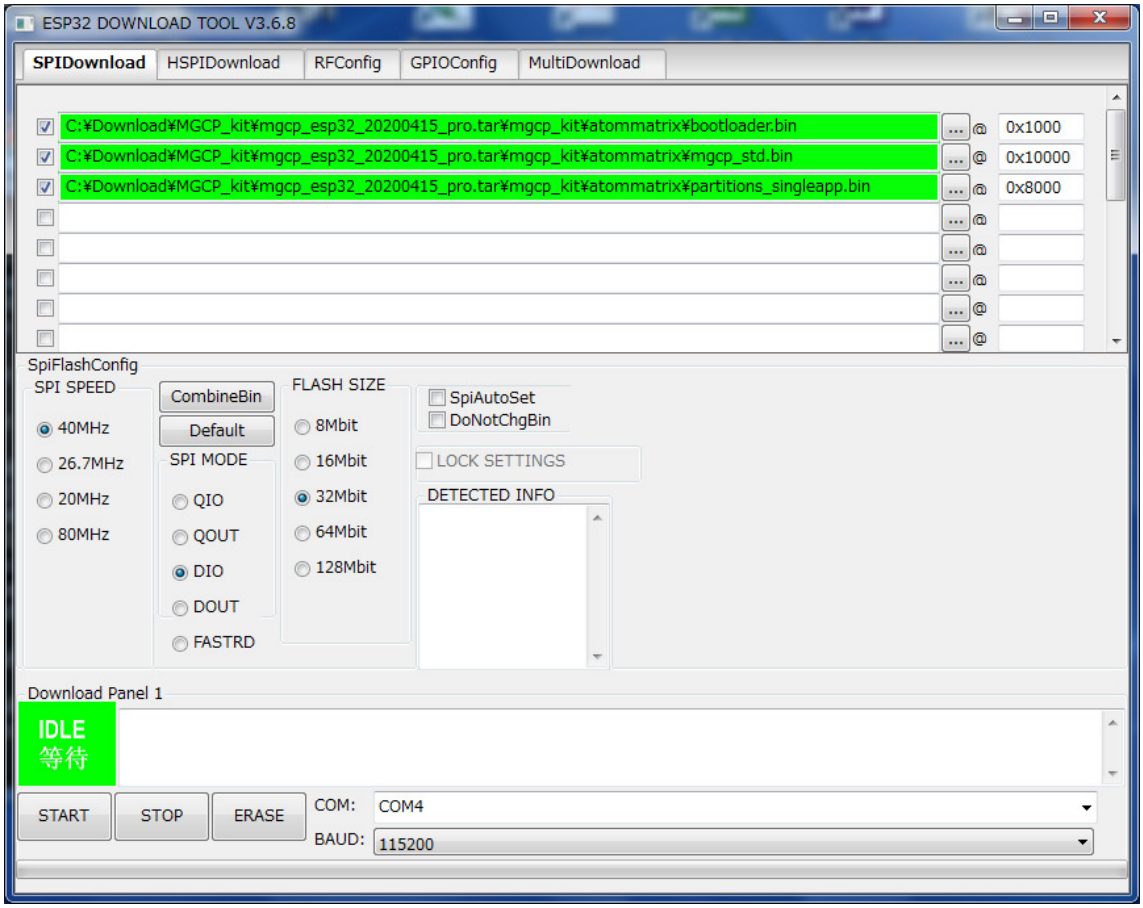
MGCP デバイスにファームウエアを書き込むために、espressif社(ESP32 プロセッサメーカ)から提供されている書き込みツール(Flash Download Tools[11])を使用します。書き込みツールを Windows コンピュータにダウンロードして起動すると下記のメニューが表示されます。



(ESP32 用ダウンロードツール起動メニュー画面)

"ESP32 DownloadTool" ボタンを選択すると下記の設定画面が表示されます。

[11] https://www.espressif.com/en/support/download/other-tools



(ESP32 DownloadTool 書き込み設定画面)

この設定画面では、先にダウンロードして展開したアーカイブファイル中のファームウエア・ファイルを選択します。"bootloader.bin","mgcp_std.bin", "partitions_singleapp.bin" の３つのファイルを指定します。それぞれのファイルの書き込みアドレスを 0x1000, 0x10000, 0x8000 に設定してください。COM 設定を MGCP デバイスが接続されているシリアルポートに変更して、BAUDをデバイスでサポートされている適切な値(115200 や 921600 等で大丈夫です)に設定してください。設定時は上記のキャプチャ画面を参考にしてください。

"START" ボタンを押すと書き込みを開始します。エラー無く書き込みが終了して "FINISH" と表示されたら "ESP32 DownloadTool" を終了します。

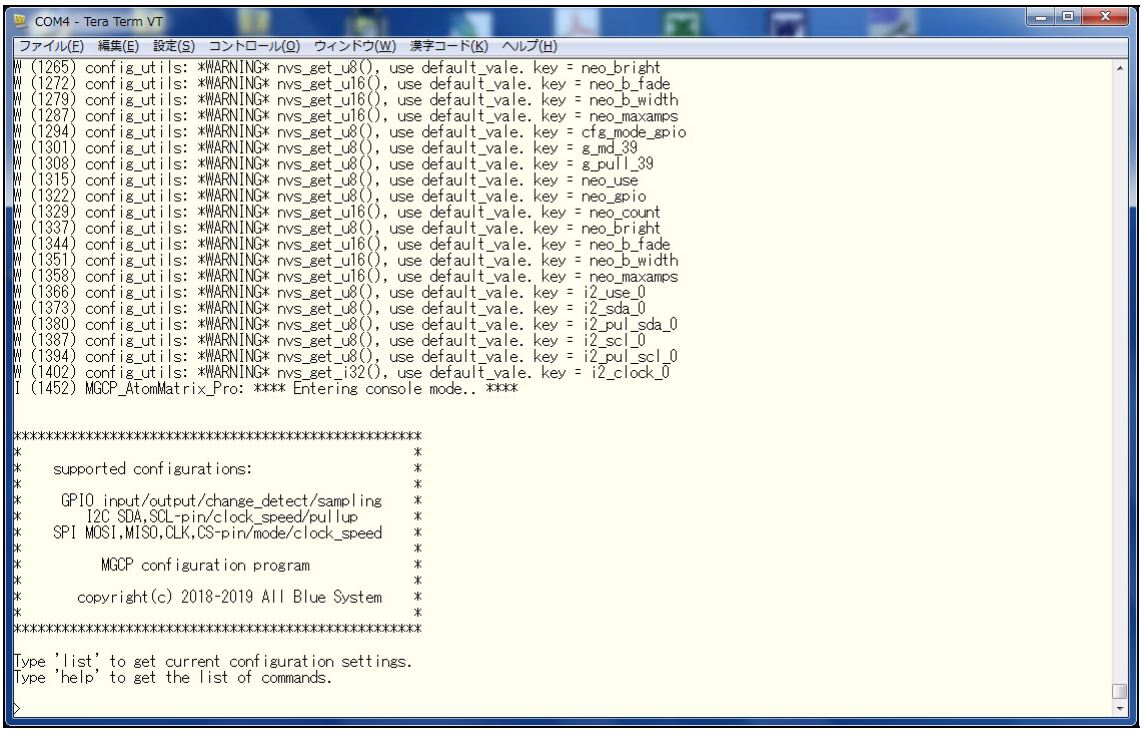
4.5 デバイス初期設定

ファームウエアの書き込みが完了したら、デバイスの初期設定を行います。初期設定作業は、MGCP デバイスのコンフィギュレーション・コンソールを Windows コンピュータ等からターミナル・エミューレータソフト等で接続して行います。ターミナル・エミューレータソフトはオープン・ソースで開発されているTeraTerm[12] 等を使用できます。

ターミナル・エミューレータソフトを起動して、接続するシリアルポートを MGCP デバイスが接続している COM ポートに設定します。また、シリアルポートの通信条件を 115200 baud, 8 data-bit, no-parity, 1 stop-bit, no-flow-control に設定します。

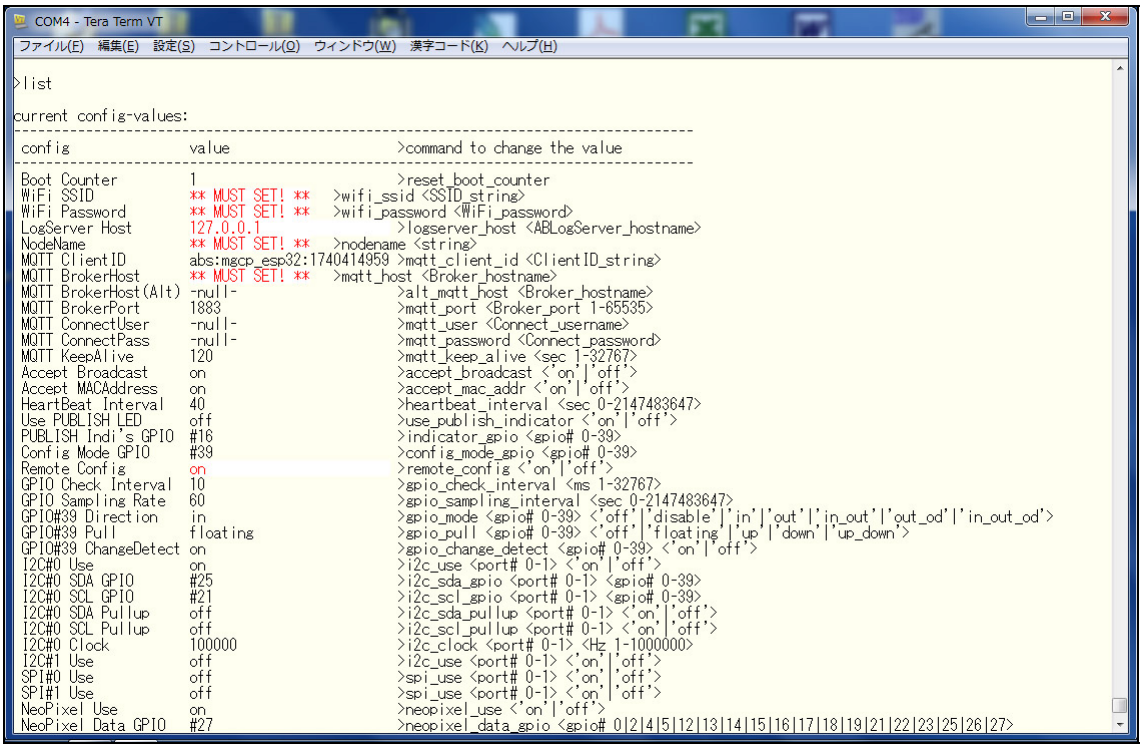
[12] https://ja.osdn.net/projects/ttssh2/

ここで指定する COM ポートはファームエア書き込み時に指定したものと同じになります。もし、ファームウエア書き込み時とは別のコンピュータに MGCP デバイスを接続する場合には、COM ポートを適切に設定して下さい。



(MGCP デバイスのコンフィギュレーション・コンソールに接続した様子)

最初のファームエア書き込み直後には、Wi-Fi 設定などの必須項目が設定されていないので必ず上記のコンフィギュレーションコンソールモードでデバイスが起動します。コンソールモードでは、デバイスタイプに合わせた初期設定値がロードされた後、コマンド入力モードになります。"list" コマンドを実行すると、現在の初期設定値が画面に表示されます。



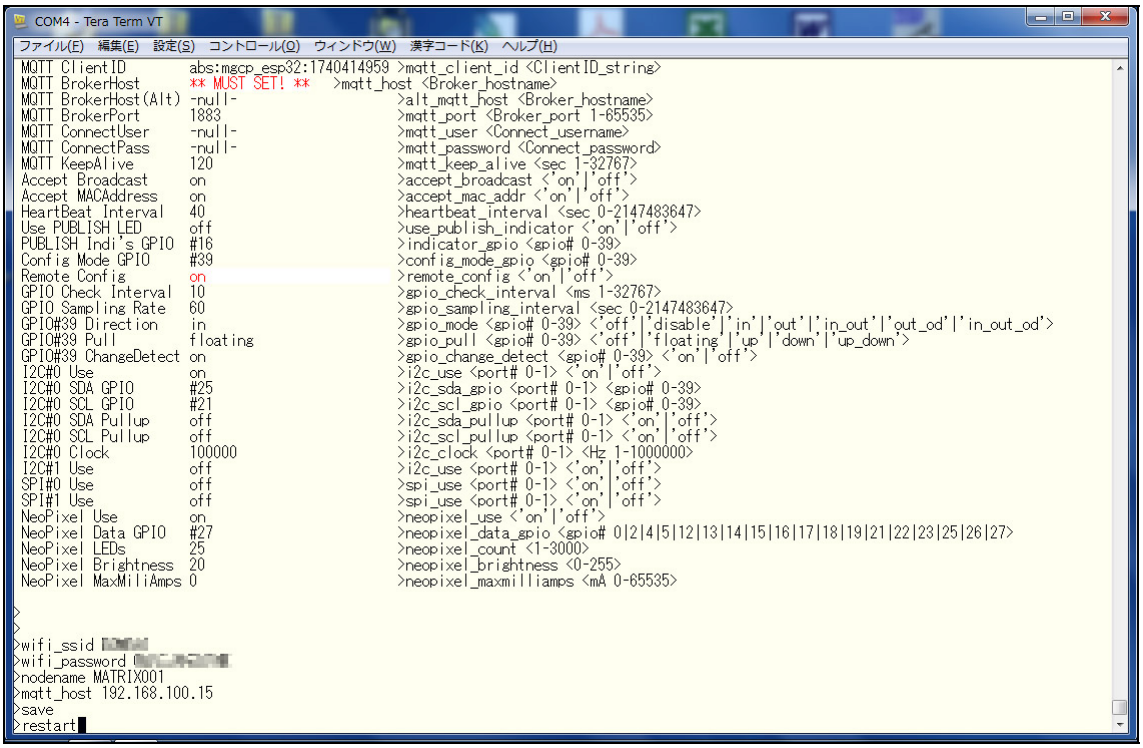
（初期設定値を "list" コマンドで確認している様子）

"list"コマンドで出力したコンフィギュレーション値が、強調表示の "** MUST SET! **" となっている項目は手動による設定が必要であることを示します。ここで、Wi-Fi とMQTT ブローカ、ノード名の設定を行います。

コンフィギュレーションコマンドの入力は下記の様になります。

```
>wifi_ssid <Wi-Fi SSID>

>wifi_password <Wi-Fi パスワード>

>nodename <任意のノード名。英数字のみ使用可能です>

>mqtt_host <MQTTブローカホスト名またはIPアドレス>

>save

>list
```

コンフィギュレーション・コマンドの詳しい説明は、このドキュメント中の "コンフィギュレーション・コマンド" の章を参照して下さい。



(Wi-Fi,ノード名,MQTT ブローカの設定を行っている様子)

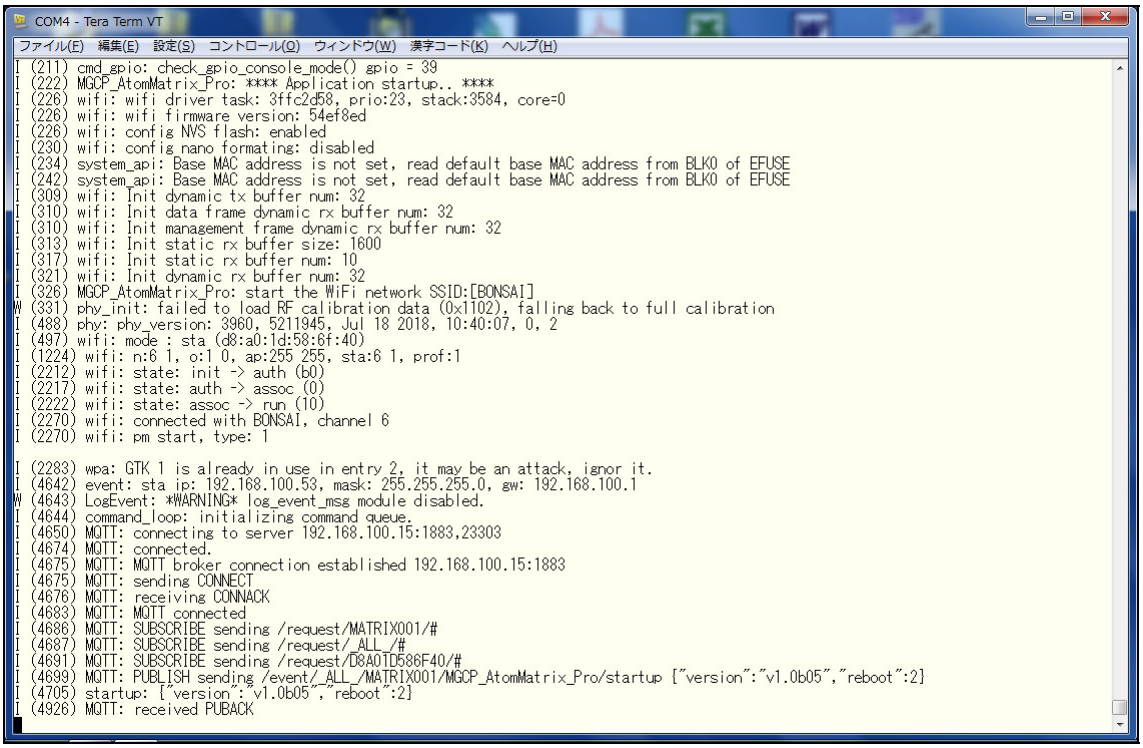
設定内容を "list" コマンドで確認した後、必ず "save" コマンドで保存してください。"restart" コマンドを実行すると、デバイスのリセットを行います。

リセット後に MGCP デバイスが Wi-Fi に接続して通常運用状態になったときには、LCD 付きの MGCPデバイス (M5Stack, M5Stick-C)では下記の様な起動画面を表示します。

(LCD付きの MGCP デバイス起動時の画面)

表示内容は上段から、ノード名、ファームウエア・バージョン、MQTT ブローカ接続先(#0:"mqtt_host"コマンドで設定したホストを使用中", #1:"alt_mqtt_host"コマンドで設定したホストを使用中")とブートカウンタです。

またこの時、シリアルポートにターミナルソフトを接続していると下記の様なログメッセージを確認できます。

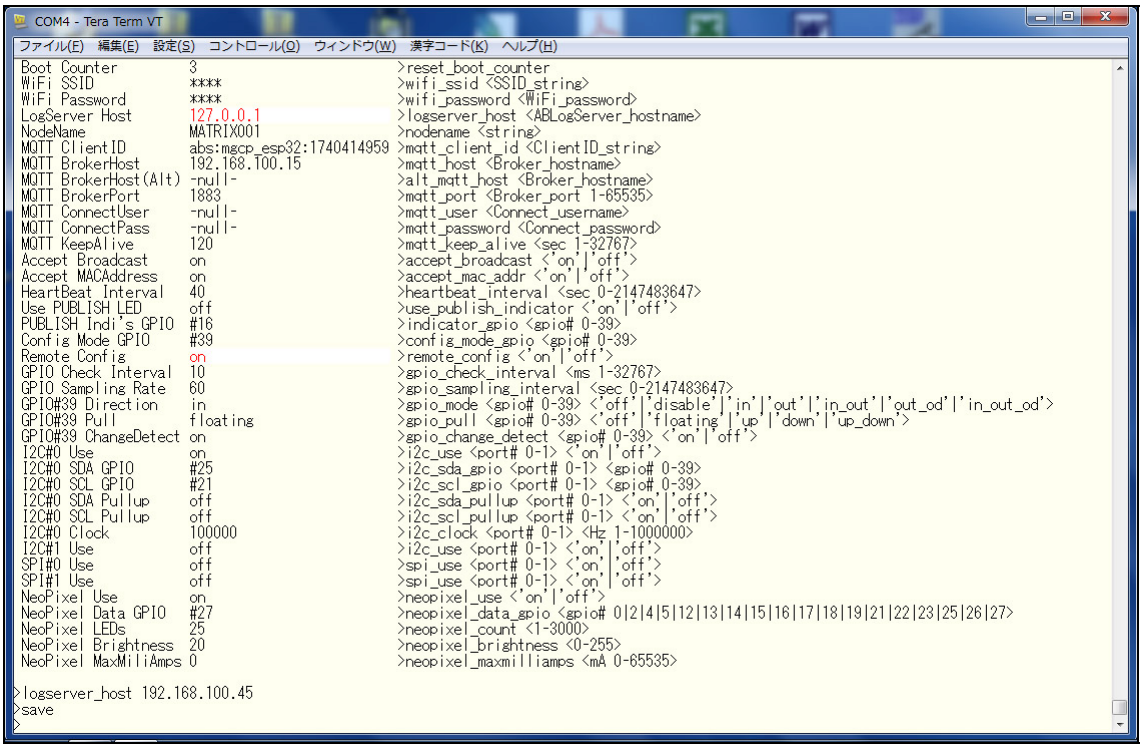


（デバイス起動メッセージがシリアルポートに出力されている様子）

通常運用状態から、追加の設定をコンフィギュレーションコンソールで行う場合には、"Config Mode GPIO" 設定(config_mode_gpio コマンド)に指定した GPIO ポートを Low にした状態でデバイスをリセットします。M5Stack, M5Stick-C, Atom-Lite,Atom-Matrix デバイスの場合には内蔵のボタン(GPIOポート)にアサインされています。

例えば Atom-Matrix デバイスの場合には正面ボタンを押した状態で、電源を OFF->ON にします。最初に設定した時と同様にコンフィギュレーション・コンソールモードでデバイスが起動しますので、追加の設定をコンフィギュレーション・コマンドで実行します。例えば、後の章で説明しているログサーバーを設置した場合の設定を行う場合には下記のコマンドを実行します。

```
>logserver_host <ログサーバーを設置した Windowsコンピュータの IPアドレス>
>save
```



（ログサーバーの追加設定を行っている様子）

I2C や SPI、Neo Pixel LED 等のデバイス設定を行う場合には、同様にコンフィギュレーション・モードに入ってコンフィギュレーション・コマンドで設定できます。またこれらのデバイス設定はリモートから MGCP コマンド経由でも可能です。詳しくは "MGCPコマンド"の章を参照して下さい。

4.6 デモライセンス使用時の制限

デモライセンスファームウエアを使用している場合には下記の制限があります。

(1) MGCPデバイス自身にパスワードを設定する "password" コマンドは使用できません。

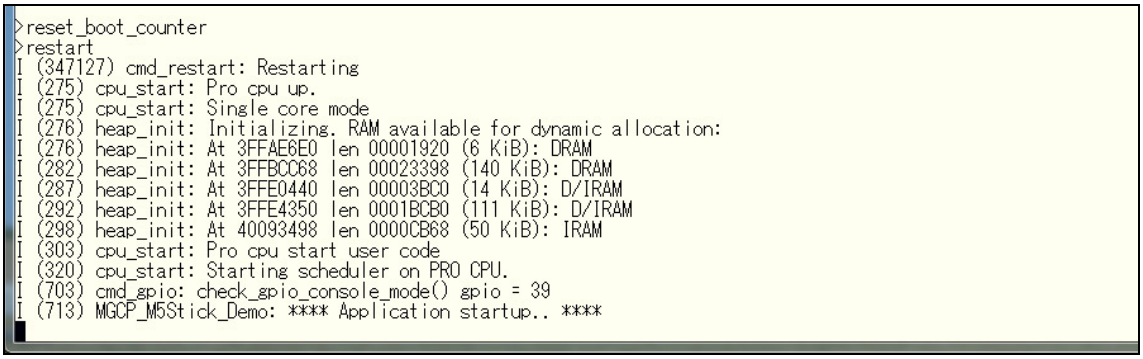
(2) abs_agent製品同梱の agent_mgcp コマンドで、リモートからMGCPデバイス全設定値の保存・更新を行うオプション指定は使用できません。またこの機能に関連する MGCPコマンド "nvs_xxxx" も無効です。

(3) MGCPデバイスのリブート(リセット) が30回以上になると強制的にコンソールモードに切り替わります。

上記(3) のリブート(リセット)の回数制限でコンソールモードに入った場合には、M5Stack/M5Stick-C の LCD ディスプレイに下記の様なメッセージを表示します。また同時に、CPUボードの GPIOポートに NeoPixel デバイスを接続している場合や、デバイス自身に NeoPixel LEDを内蔵している "Atom-Matrix" や "Atom-Lite"デバイスでは、LED#0 が赤色に点滅します。

（デモライセンスのブートカウントエラーが表示されている様子）

上記の表示になっている場合、MGCP デバイスはコンフィギュレーションモードで起動しています。シリアルターミナルソフトでコンフィギュレーションコンソールに入って "reset_boot_counter" コマンドを実行することで再びデモライセンスで使用を継続することができます。



（デモライセンスを有効にするためにブートカウンタをクリアしている様子）

5 ログサーバー・インストール(オプション)

MGCP デバイスは実行時に発生するコマンド実行結果、イベント、ハードウエアエラーやその他のシステムメッセージ等をログに出力する機能があります。ログサーバーを設置しない場合でも MGCP デバイスの動作に影響はありませんので、ログサーバーの設置は任意オプションになります。

ログの出力先は MGCP デバイスのコンフィギュレーション・コンソール(後述) 中の "logserver_host" コマンドで指定したログサーバーに出力します。デフォルトではループバック・ホスト(127.0.0.1) が指定されていてログ出力は無効になっています。"logserver_host" コマンド設定値を ABS-9000 LogServer プログラムを設置した Windows コンピュータの IPアドレスもしくはホスト名に指定すると、MGCP デバイスで出力したログメッセージを収集・保管したり、リアルタイムにメッセージを確認することができます。

複数の MGCP デバイスで出力するログを１つのログサーバーに出力することもできます。オールブルーシステムの abs_agent製品 や ABS-9000 DeviceServer 製品のログも MGCP デバイスと共通のログサーバーに出力できますので、全てのログを集中して管理することができるようになります。

5.1 ログサーバー動作環境

ログサーバーのインストールには ABS-9000 LogServer インストールキットを使用します。ABS-9000 DeviceServer のインストールキットを使用して、ログサーバー部分を併用することもできます。MGCP デバイスまたは abs_agent 製品のみを運用する場合には、ログサーバーとログコンソールプログラムのみをインストールする ABS-9000 LogServer インストールキットを使用する方が便利です。このマニュアルでは、ABS-9000 LogServer インストールキットを使用したインストール方法について説明します。ABS-9000 LogServer はフリーでインストールして使用することができます。

ログサーバーは、"ABLogService" の名前が付いた Windows サービスプログラムと、ログコンソールプログラム（Win32アプリケーション）の２つで構成されています。

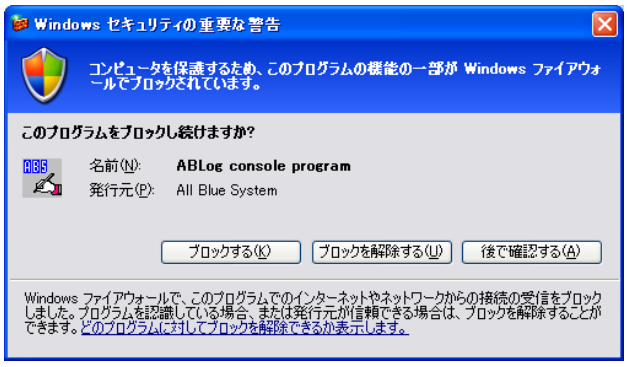
ログサーバーの動作環境は以下になります。インストール前に動作環境を満たしていることを確認してください。ここで示した Windows 以外の最新版 Windows OS で正常に動作可能と思いますが、これらの OS にインストールする場合には、Windowsの "UAC"（ユーザー・アカウント制御）機能を無効に設定してからインストールしてください。（詳しくは後述）

項目	必要なバージョン・リソース
オペレーティングシステム	Windows 2000 Professional Windows XP (32bit) SP2 または SP3 Windows 2003 Server (32bit) Windows 7(64bit) SP1
CPU クロックスピード	2GHz 以上
メインメモリ	1Gbytes 以上
ディスク容量（Cドライブ下の "Program Files"フォルダ）	150Mbytes 以上の空き容量
ネットワークポート	10Mbps/100Mbps イーサネットポート１つ以上

MGCP デバイスなどログメッセージ送出側から、ログサーバーが動作しているコンピュータへ安定してネットワーク・アクセスするために、ログサーバーをインストールする Windows PC には必ず LAN 内の固定 IPアドレスをアサインしてください。

5.2 ログサーバーで使用するポート番号をWindows で利用可能にします

ログサーバーとログコンソールプログラムでは下記の表で示したネットワークポートを使用しています。ファイアウォールプログラムやセキュリティソフト等を使用している場合は、abs_agent 側からログメッセージを受信するために、Windows PC 側の設定変更が必要になります。ここで設定しない場合には、ログコンソール起動時や ABS-9000 LogServer インストール作業中に下記のようなダイアログが表示される場合があります。



この画面が表示された場合には、”ブロックを解除する” を選択することで該当するポートが使用可能になります。ただし、セキュリティ上 ABS-9000 LogServer インストールキット起動前に手動でポートを使用可能にしておくこと

をお勧めします。全てのポートを正しく設定した場合には上記のダイアログは表示されません。

ログサーバーとログコンソールプログラムが使用するプロトコルとポート番号		
プログラム・機能	**プロトコル**	**ポート番号(名称)**
ログサーバーのメッセージ送受信	UDP	2056 2057

ここでは、Windows XP と Windows 7のファイヤーウォール設定について説明します。他の OS や市販のセキュリティソフトを使用している場合には、それらのソフトウエアマニュアルを参照して同様のポート開放の設定を行ってください。

⚠市販のセキュリティソフトを使用している場合にはポート番号を開放して外部から接続可能にして下さい

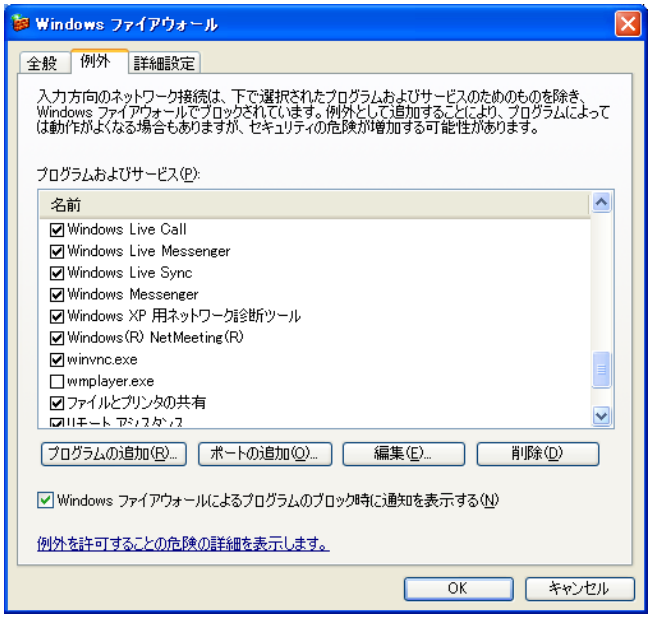
ログサーバーはネットワーク経由でアプリケーションやデバイス等からアクセスされますので、上記のポート番号を外部から接続可能に設定してください。セキュリティソフトがアクセス可能にする対象の、プログラム名を必要とする場合には

"C:¥Program Files¥AllBlueSystem¥ABLogServer.exe" (Windows XP) または、

"C:¥Program Files (x86)¥AllBlueSystem¥ABLogServer.exe" (Windows7 64bit) を指定してください。

5.2.1 Windows XP(SP3) でポート開放操作

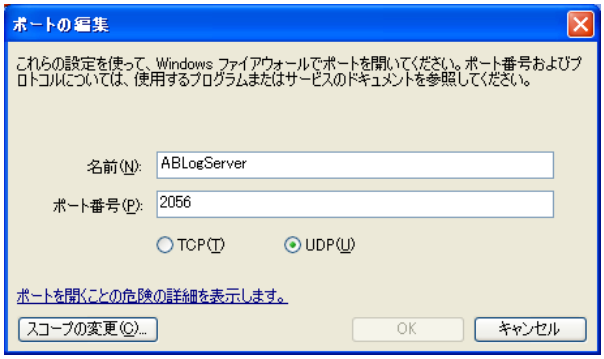
コントロールパネルから Windows ファイヤーウォールを選択して、"例外" タブを選択します。



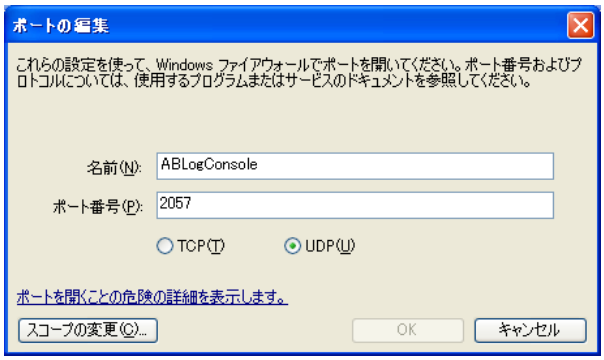
(Windows ファイヤーウォール 設定画面 Windows XP SP3 の場合)

"ポートの追加" を押して、ログサーバーで使用するポート 2056/UDP の設定を追加して、

"OK" を押してください。ポートの編集ダイアログで記入する "名前" には、他の文字列を設定しても構いませんが、ポート番号と TCP/UDP の区別は間違えないように注意して下さい。



ログサーバーのポート 2056/UDP の設定を追加します。（プロトコルが UDP になっている点に注意してください）

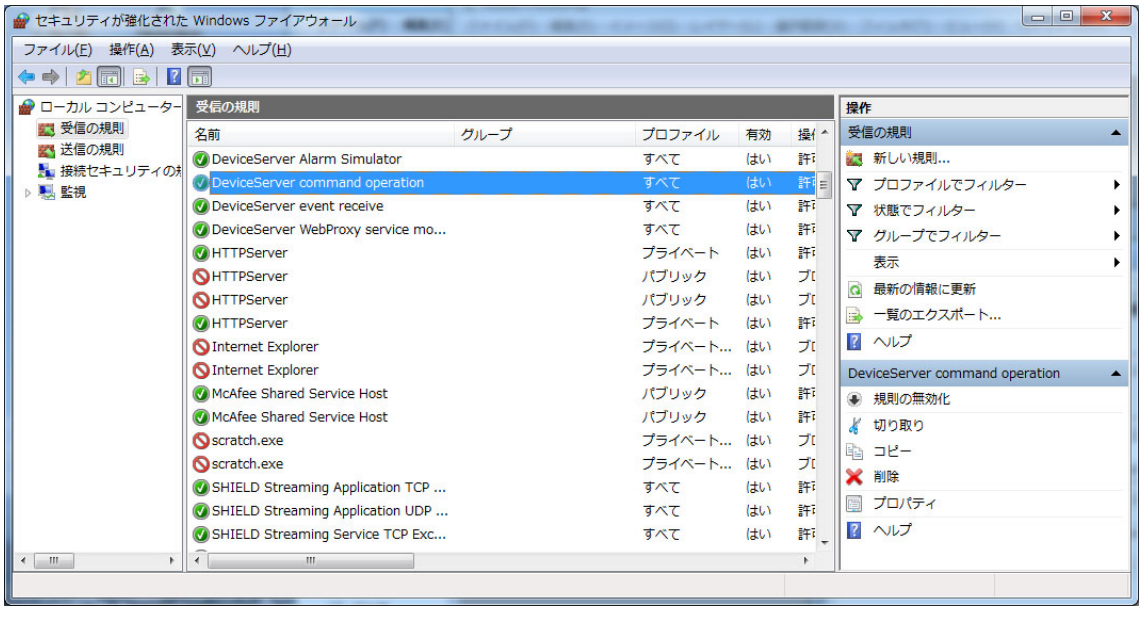


ログコンソールのポート 2057/UDP の設定を追加します。（プロトコルが UDP になっている点に注意してください）

以上でファイヤーウォールの設定は完了です。

5.2.2 Windows 7(SP1) でポート開放操作

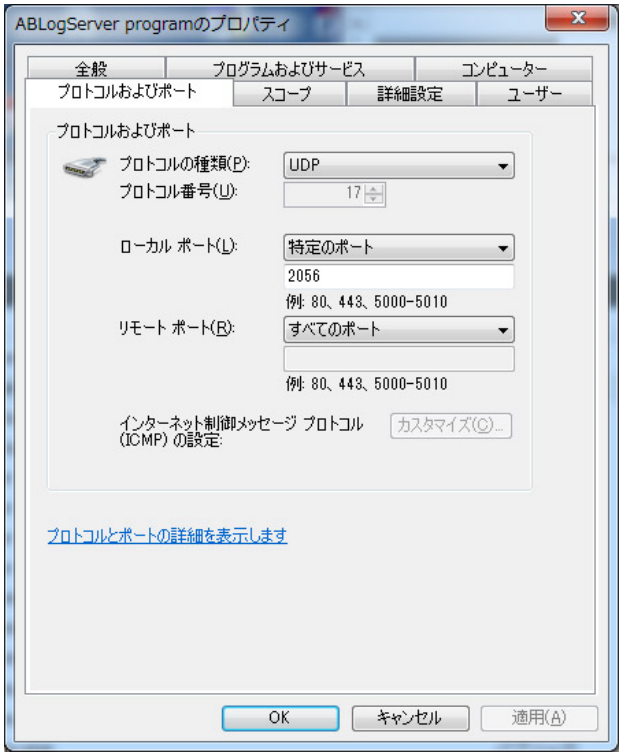
Windows 7 の場合にも同様にコントロールパネルから Windows ファイヤーウォール設定画面を表示します。詳細設定の中の "受信の規制" を選択してポートの開放操作を行います。



(Windows ファイヤーウォール 設定画面 Windows7 SP1 64bitの場合)

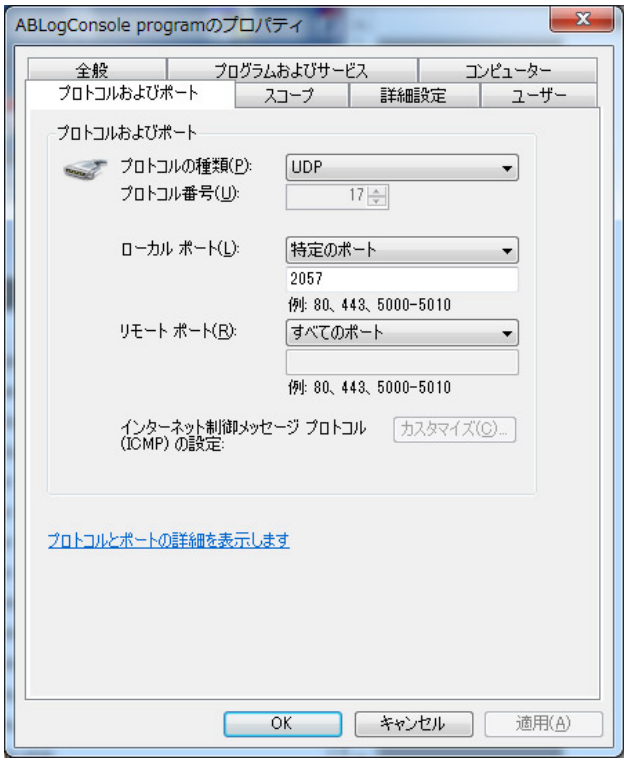
ファイヤーウォール画面の "新しい規制"を選択してウィザード形式で作業を行います。"ポート" を選択して TCP/UDP のポート番号の規制を作成する画面を出して Windows XP の場合と同様に UDP プロトコルの 2056と 2057 を開放します。

最初にローカルポート設定欄にポート番号(2056) を指定して、リモートポート設定欄には"全てのポート" を選択します。規制の名前は "ABLogServer program" にしていますが適当に付けても構いません。



(UDP/2056 ポートの開放を行う設定例 Windows7 SP1 64bitの場合)

次に"新しい規制"を選択して同様にポート番号(2057) の項目を追加します。この例では規制の名前に "ABLogConsole program" を指定しています。



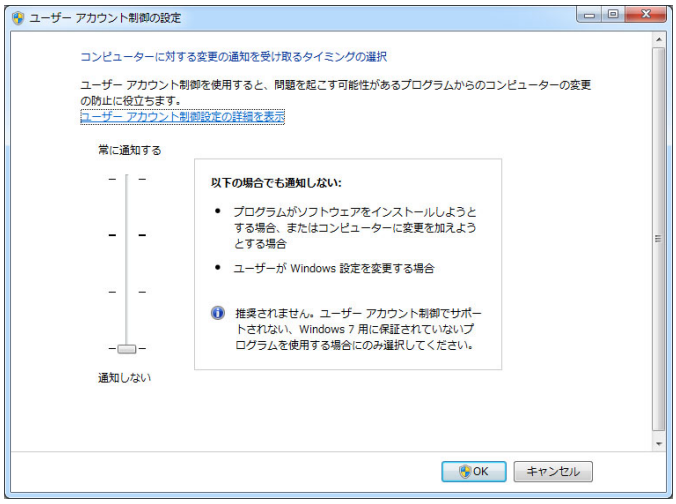
(UDP/2057 ポートの開放を行う設定例 Windows7 SP1 64bitの場合)

全てのログサーバー用のポート開放作業が終了したときには、規制の一覧に下記の項目が追加されているのを確認してください。

(Windows ファイヤーウォール設定画面の、ログサーバー動作用に追加した受信規制項目一覧)

5.3 ログサーバーの Windows UACを無効にする

ログサーバー は Windows2000,WindowsXP SP3、Windows2003 Server, Windows 7 SP1(64bit) の環境で動作試験をしていますが、その他の最新バージョンの Windows でも動作させることができます。OS のバージョンによって設定方法は異なりますが、コントロールパネル等から**UAC のアクセス制御を完全に "無効"(通知しない) に設定した後、OS を再起動する**ことで最新バージョンの Windows でも使用できます。



(Windows7 のユーザーアカウント制御の設定で "通知しない"に設定している様子)

これは、ABS-9000 DeviceServerインストールキットまたは、ABS-9000 LogServerインストールキットでインストールされる "ログコンソールプログラム" が、Windows の SCM (サービスコントロールマネージャ) 機能を使用しているためです。"ログコンソールプログラム" 起動時に、ユーザーアカウントの警告ダイアログが出力される事があります。この警告ダイアログが表示された場合には、上記の設定を行った後もう一度 "ログコンソールプログラム" や "サーバー設定プログラム" を起動してください。一度設定すればこのダイアログは表示されなくなります。

その他の ログサーバー等のサービスプログラムは Windows7 で完全に動作します。お客様がご使用中の Windows 環境で、セットアップ時や運用中に問題が発生した場合には下記のメールアドレス宛にご連絡下さい。
問い合わせメールアドレス
contact@allbluesystem.com

5.4 ログサーバーインストール例

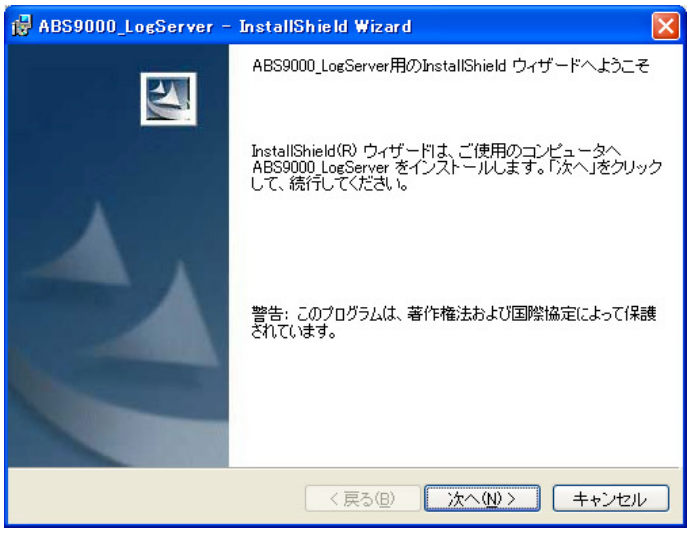
ここからは、ログサーバーに使用する Windows PC に ABS-9000 LogServer のインストールキットを使用してインストール作業を行います。

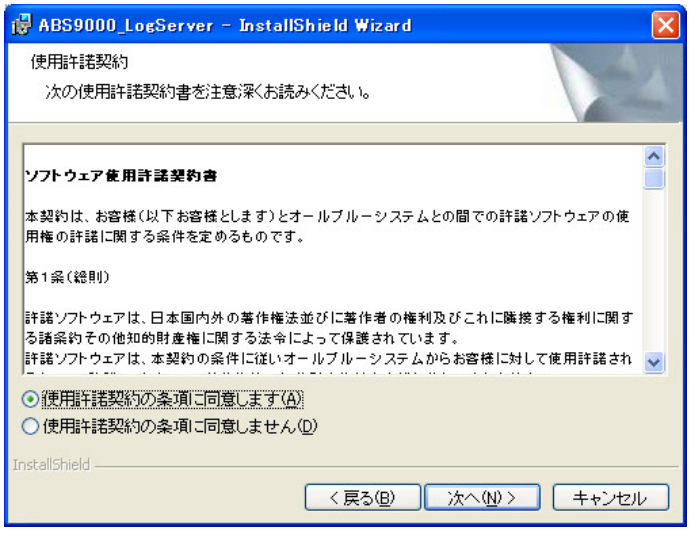
インストール時に "ABLogService" の名前が付けられた Windows サービスプログラムを登録します。全てのセットアップはウィザード形式で行われます。インストール中に表示される設定項目は、全てデフォルト値で設定できますので、特に設定内容を変更しなくてもクリックを押すだけでインストールが完了します。

インストールキットを実行するときは、Windows のシステム管理者権限のユーザー(Administrator等)で行います。Windows 一般ユーザーの権限ではセットアップできませんので注意してください。

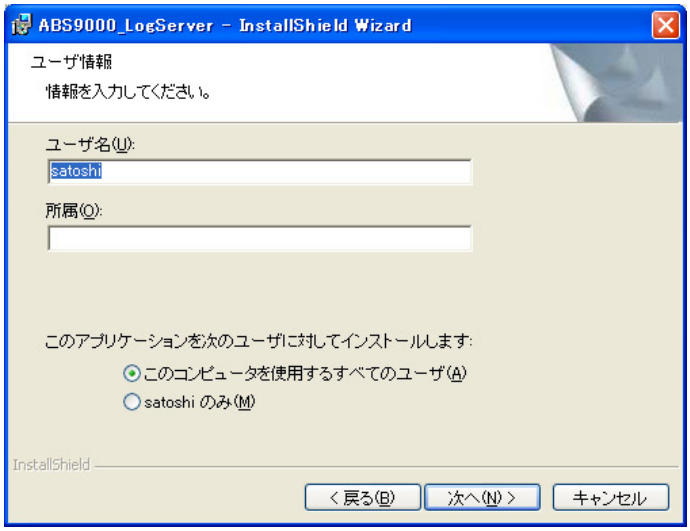
インストールキットは ZIP 形式で１つのファイルにまとめられていますので、適当なディレクトリにファイルを展開してください。WindowsXP, Windows2003, Windows7 の場合にはファイルを右クリックして “全て展開” メニューでファイルを展開することができます。ファイルを展開したら、”setup.exe” をダブルクリックしてインストーラプログラムを起動します。

以下は Windows XP(SP3) にインストールするときの表示例です。他の Windows で作業するときも操作は同一です。

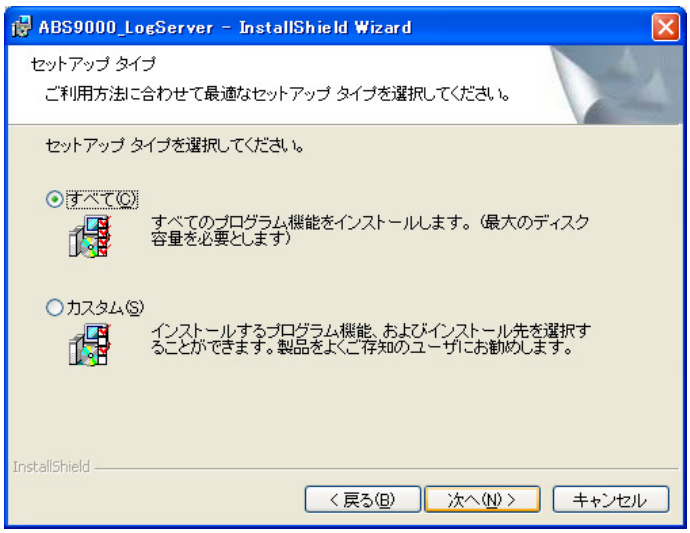




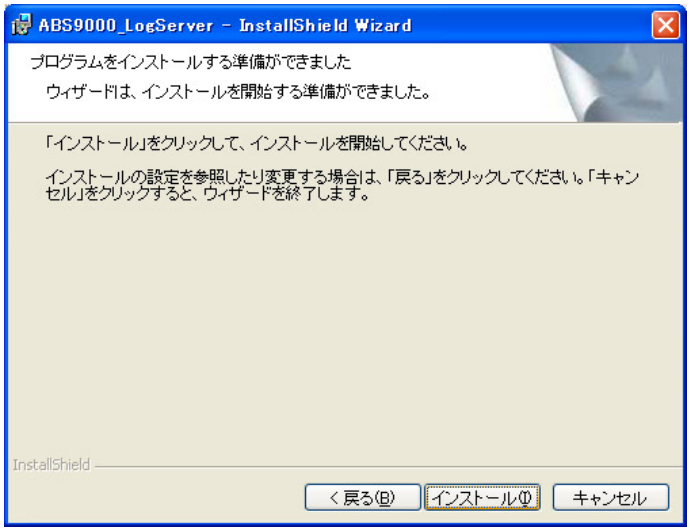
使用許諾をよく読んだ上で、同意するにチェックを付けてください。同意されない場合は、ABS-9000 LogServer を使用することはできません。



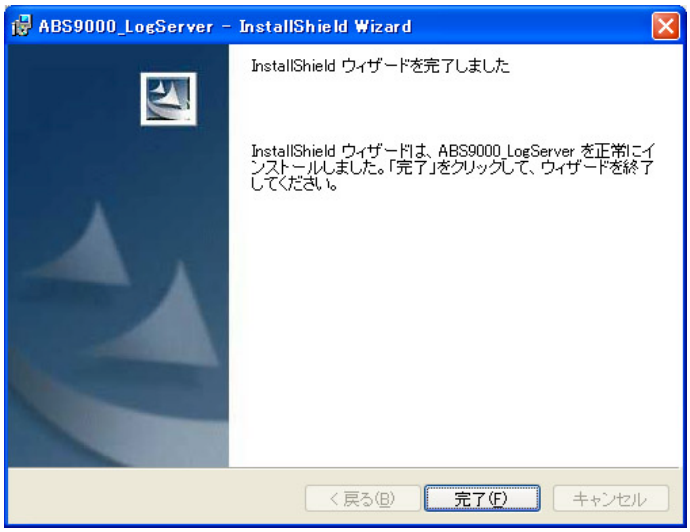
LogServer はデフォルトの ”このコンピュータを使用するすべてのユーザ “を選択してください。ユーザー名や所属などは変更する必要はありません。



セットアップタイプはデフォルトの “すべて”を必ず選択してください。デフォルトのインストールフォルダを変更した場合や、一部のコンポーネントのみのインストールを選択すると正常に動作しません。

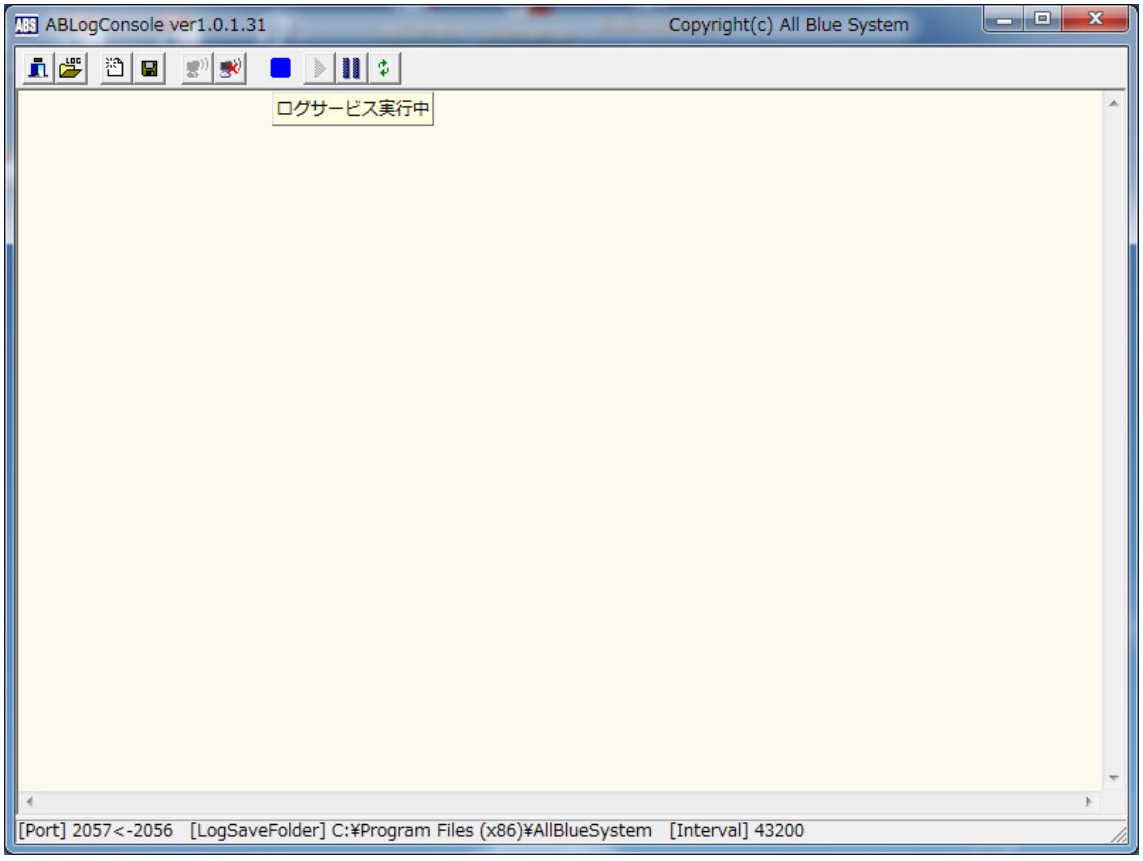


インストール準備が完了しましたので “インストール”を押してください。



これで、Windows PC 上にログサーバーの設置が終了しました。ログサーバーはサービスプログラムで実行していますので、Windows PC が起動している間は常にログメッセージを収集して内部で保管しています。

プログラムメニューから "All Blue System" -> "ログコンソール" を選択してコンソールプログラムを起動します。ログコンソールプログラムを起動すると、現在ログサーバーで取得中のメッセージを画面に表示することができます。



（ログコンソールプログラムを実行している様子）

ログコンソールプログラムから、ログサーバーで保存している過去のログファイルを参照することもできます。また、現在ログサーバー中でメモリ中にキャッシュされているログメッセージを強制的にファイル出力させる機能もあります。ログコンソールプログラムを終了してもログサーバープログラムは常にサービスプログラムとして実行中ですので、ログメッセージが失われることはありません。詳しい機能は後述します。

5.5 ログサービス(ABLogServer)機能

オールブルーシステム製の abs_agent, MGCP, DeviceServer等で出力した全てのログ情報を記録します。
クライアントプログラムの幾つかは直接ログサービスにイベントを記録するものがあり、これらも同様に記録します。

ログサービスは ABS-9000 LogServer をインストールしたWindows コンピュータ上のサービスプログラムとして動作しています。ユーザーが直接このサービスプログラムを操作することはありません。アプリケーション(abs_agent, MGCP, DeviceServer等)で検出したエラーやイベント、クライアントアクセスの記録、ユーザーが作成したスクリプトからのログ出力を、集中管理して記録しています。

ログサービスは記録動作を高速に行うために、通常はメモリ中にログ情報を保管していて、定期的にファイルに書き出す方法を採用しています。定期的に出力されたログファイルは、いつでもユーザーが参照できます。後述のログコンソールプログラムを起動することで、リアルタイムにログ出力を画面に表示することや、ログサービスに対してメモリ中に保管中のログを強制的に、ファイル出力するように指示することもできます。

Windows サービス名	ABLogService
プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABLogServer.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogServer.exe
設定ファイル名	C:¥Program Files¥AllBlueSystem¥ABLogService.ini Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogService.in
バージョン番号の確認方法	プログラムファイルのプロパティを開いて、バージョン情報タブ内のファイルバージョンを確認する
ログファイル出力フォルダ [Folder]	C:¥Program Files¥AllBlueSystem¥Logs Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥Logs
ログファイル名	“abs” + *YYYYMMDDHHMMSS* + “.log” ファイル作成時のタイムスタンプをファイル名に利用する。
ログファイル切り替えタイマー間隔 [Interval]	約 6 時間（設定ファイルの単位は秒で指定される） ログコンソールプログラムから強制的にファイルを切り替えることもできる。またログサービス再起動時も自動的に切り替えが行われる。
ログイベント受信ポート(UDP) [ReceivePort]	2056
ログイベント送信ポート(UDP) [RepeatPort]	2057 ログコンソールプログラムでリアルタイムにログ情報を表示するために使用する。
ログファイル保管期間 [LogKeepDays]	30 日 保管期間を過ぎたログファイルは自動的に削除される

上記の設定項目の幾つかは、設定ファイル(ini ファイル)を修正することで、設定値を変更することができます。”[xxxx]” 部分が対応する設定ファイルのタグ名です。詳しくは設定ファイルの内容を参照してください。（ただし、ログイベントポート番号の変更は行わないで下さい。サーバーからのログを受信できなくなります）設定を変えた場合はログコンソールプログラムからログサービスを再起動するか、PC を再起動することで新しい設定内容が有効になります。

設定ファイルの内容(例)

```
[ABLogService]
ReceivePort=2056
RepeatPort=2057
Interval=21600
Folder=C:¥Program Files¥AllBlueSystem¥Logs
LogKeepDays=3
```

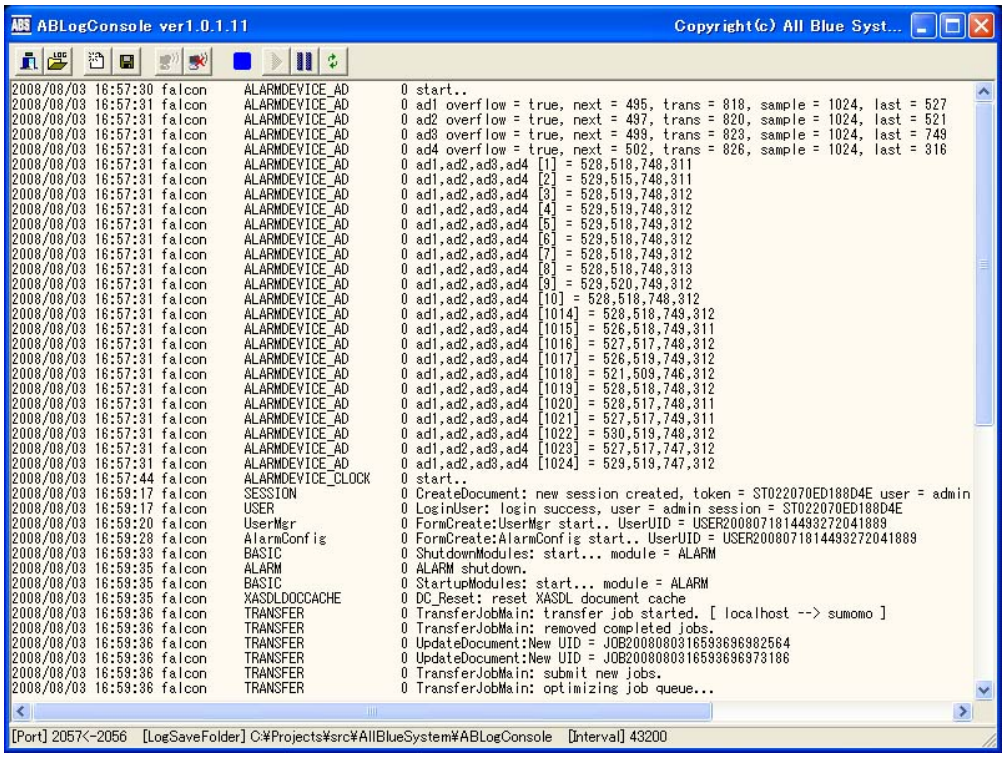
5.6 ログ管理コンソール(LogConsole)機能

ABS-9000 LogServer の動作するPC で起動させて、ログサービス(前述) に記録しているイベントを画面上でリアルタイムに確認することができます。

ログ管理コンソールは何時でも起動することが可能で、そのときに発生しているログイベントをログサービスから取得して表示します。ログ管理コンソールを終了してもログイベントは常にログサーバーで収集・保管していますので、ログが失われることはありません。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABLogConsole.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogConsole.exe
バージョン番号の確認方法	プログラム起動時のメインフォームタイトルに表示される

プログラムを起動すると自動的にログサーバーで収集している現在のログメッセージを画面に表示し続けます。



各ツールボタンの機能を以下に説明します。

ツールボタン	説明
アプリケーション終了 	ログ管理コンソールプログラムを終了する。 ログイベントの記録はログサービスで行われるので、ログが失われることはない。
ログフォルダを開く 	ログサービスで保管されたログファイルを参照する。 ボタンを押すとエキスプローラでフォルダが展開される。過去に保存されたログをここから選択してワードパッドやメモ帳などで表示してください。
メッセージクリア	画面に表示されたログメッセージをクリアする。

	
メッセージセーブ 	画面に表示されたログメッセージをファイルに保存する。 ログサービスで自動保管されるログファイルとは別に、ログ管理コンソールの画面上にあるメッセージのみをユーザーの指定したファイルに保存するために使用する。
ネットワーク接続 	ログ管理コンソールでログメッセージを受け付ける状態にする。 （起動時は接続済のため、選択できないようになっています）
ネットワーク切断 	ログ管理コンソールでログメッセージを受け付けない状態にする。 ログ管理コンソールの画面上のログ更新を一時的に停止したい場合に使用する。切断中のログは破棄されるので、その間のログを後から確認したい場合は、ログサービスで自動的に保管されたログファイルを参照してください。
ログサービス開始 	ログサービスを開始する。 （通常は、サービスは常に動作中のため選択できないようになっています）
ログサービス停止 	ログサービスを停止する。 ログサービスの設定ファイルを修正したい場合等に、サービスを停止するために使用します。
ログファイル切り替え （ログファイルフラッシュ） 	ログサービスの出力ファイルを強制的に切り替える。 このボタンを押すと、ログサービスで予め設定されたログファイル切り替えタイマー間隔(デフォルトで 6 時間) を無視して、強制的にファイル切り替えが行われます。 現在メモリ中にあるログメッセージを直ぐに確認したい場合に使用します。

6 アンインストール・アップデート

6.1 MGCP ファームウエアアップデート

MGCP デバイスのファームウエア更新(アップデート)は、新規インストールと全く同じ手順で行います。

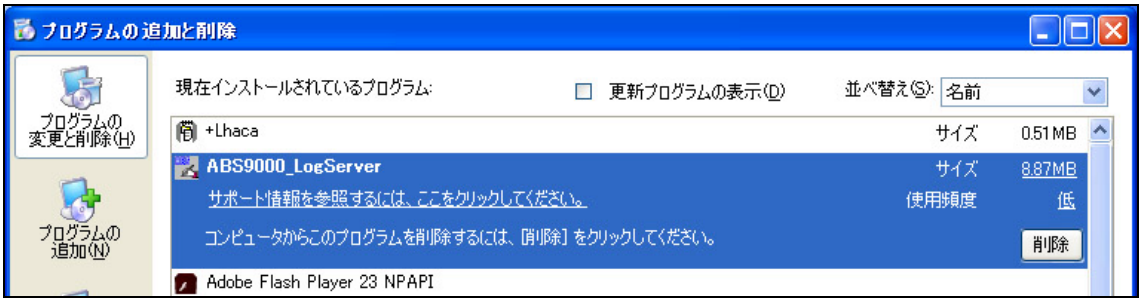
更新時にはデバイス中の不揮発領域(NVS)に保存されている既存の設定情報をそのまま引き継いで使用します。このため、ファームウエア書き込み後のデバイス初期設定作業は不要でリセット後から直ぐに使用できます。

詳しい作業手順は "MGCP デバイス・インストール"の章を参照してください。

6.2 ログサーバーアンインストール

ログサーバーを削除する場合には、設置した Windows PC 上のコントロールパネルから、"プログラムの追加と削除"で "ABS-9000 LogServer" を指定して削除してください。アンインストーラでは、ログサーバーのサービスプログラムも自動的に削除します。

Windows のシステム管理者権限のユーザー(Administrator等)でWindows にログインした後、Windows コントロールパネルの"プログラムの追加と削除"から “ABS9000_LogServer” を選択して “削除” を押します。



7 MGCPイベント

7.1 startup(ファームウエア起動)

MGCP デバイスが起動したときに最初に一度だけ送信します。デバイスが何らかの原因によって再起動した場合にも同様に送信します。

event topic

/event/_ALL_/<node_name_from>/<application_name>/startup

<node_name_from> には MGCP デバイスにユーザーが設定したノード名文字列が入ります。

<application_name> は MGCP デバイスで使用しているファームウエア種別を示す文字列が入ります。

例：/event/_ALL_/M5_NO01/MGCP_M5Stack_Pro/startup

event data

{"version":<version_string>,"reboot":<boot_count>,"tx_power":<power>}

<version_string> にはMGCPファームウエアのバージョン番号を示す文字列が入ります。

<boot_count> には MGCPデバイスの起動回数を示すカウンタ値(整数) が入ります。

<power> には Wi-Fi 最大送信強度を示す数値が入ります。(数値の目安 34 -> 8.5dBm, 78 -> 19.5dBm)

例：{"version":"v1.0b06","reboot":215,"tx_power":34}

7.2 heartbeat(定期的なデバイス情報の送信)

コンフィギュレーション・コマンド "heartbeat_interval" で設定した間隔(デフォルトは 40秒)で、MGCP デバイス情報を送信します。

event topic

/event/_ALL_/<node_name_from>/<application_name>/heartbeat

<node_name_from> には MGCP デバイスにユーザーが設定したノード名文字列が入ります。

<application_name> は MGCP デバイスで使用しているファームウエア種別を示す文字列が入ります。

例：/event/_ALL_/ESP32_NO1/MGCP_STD_Pro/heartbeat

event data -- default設定

{"reboot":<boot_count>,"free_heap":<free_heap_size>,"mqtt_restart":<broker_connect_count>,"mac":<MACAddress>,"ip":<IPAddress>}

event data -- ダイレクトコマンド用の UDP サーバー有効時(dx_command = 1)

{"reboot":<boot_count>,"free_heap":<free_heap_size>,"mqtt_restart":<broker_connect_count>,"mac":<MACAddress>,"ip":<IPAddress>,"dx_port":<DXPort>}

<boot_count> には MGCPデバイスの起動回数を示すカウンタ値(整数) が入ります。
<free_heap_size> には MGCP デバイスで利用可能なフリーメモリサイズ(bytes)が入ります。
<broker_connect_count> には、MQTT ブローカに接続した回数が入ります。何らかの原因で MQTT ブローカとの接続が切断した場合には、MQTT デバイスは自動的に再接続を試みます。このときにこのカウンタ値がインクリメントされます。Wi-Fi 接続が切断された場合には MGCPデバイス自身をリセットして再起動しますので、このカウンタ値はクリアされます。
<MACAddress> には MGCPデバイスの物理アドレスが 16進数表記の文字列で入ります。
<IPAddress> には MGCPデバイス起動時に DHCP サーバーからアサインされた IPアドレスが文字列で入ります。このIPアドレスは MGCPデバイスが設置されているローカル(LAN)内でのみ有効です。
<DXPort> にはダイレクトコマンド用 UDPサーバーを有効にしている場合の IP ポート番号が入ります。

例：{"reboot":98,"free_heap":155944,"mqtt_restart":1,"mac":"240AC4077F74","ip":"192.168.100.50"}

7.3 gpio_change_detect (監視中のGPIO値が変化)

コンフィギュレーション・コマンド "gpio_change_detect" もしくは MGCP コマンド "gpio_change_detect"で監視対象に設定した GPIO ポート値が変化した場合にイベントを送信します。

event topic

/event/_ALL_/<node_name_from>/<application_name>/gpio_change_detect

<node_name_from> には MGCP デバイスにユーザーが設定したノード名文字列が入ります。
<application_name> は MGCP デバイスで使用しているファームウエア種別を示す文字列が入ります。

例：/event/_ALL_/STICK001/MGCP_M5Stick_Pro/gpio_change_detect

event data

{"seq":<sequence_number>,"gpio_val":<gpio_value>","gpio_diff":<gpio_changed>}

<sequence_number> には MGCPデバイスがこのイベントを送信した回数を示すカウンタ値(整数) が入ります。
<gpio_value> にはイベント送信時の、MGCP デバイスの全GPIO 値が 16進数文字列で入ります。ビット位置が GPIO 番号と一致していて、そのビット値が 1 の場合には High, 0 の場合には Low を示します。また、<gpio_value>は各GPIO ポートに設定されているモードに関係なく強制的に全 GPIO 値を取得します。
<gpio_changed> には変化したGPIOポートを示す値が16進数文字列で入ります。ビット位置が GPIO 番号と一致していて、そのビット値が 1 の場合には変化したことを表し, 0 の場合には変化していないかもしくは監視対象外であることを示します。

例 : {"seq":1384,"gpio_val":"A000634A8B","gpio_diff":"1000000000"}

7.4 gpio_sampling (定期的なGPIO値取得)

コンフィギュレーション・コマンド "gpio_sampling_interval" で指定した間隔で GPIO ポート値を取得してイベントを送信します。

event topic

/event/_ALL_/<node_name_from>/<application_name>/gpio_sampling

<node_name_from> には MGCP デバイスにユーザーが設定したノード名文字列が入ります。
<application_name> は MGCP デバイスで使用しているファームウエア種別を示す文字列が入ります。

例 : /event/_ALL_/STICK001/MGCP_M5Stick_Pro/gpio_sampling

event data

{"seq":<sequence_number>,"gpio_val":<gpio_value>"}

<sequence_number> には MGCPデバイスがこのイベントを送信した回数を示すカウンタ値(整数) が入ります。
<gpio_value> にはイベント送信時の、MGCP デバイスの全GPIO 値が 16進数文字列で入ります。ビット位置が GPIO 番号と一致していて、そのビット値が 1 の場合には High, 0 の場合には Low を示します。また、<gpio_value>は各GPIO ポートに設定されているモードに関係なく強制的に全 GPIO 値を取得します。

例 : {"seq":1,"gpio_val":"A000634A8B"}

7.5 job_exec (定期的なダイレクトコマンド実行)

MGCP コマンド "job_config" で設定したダイレクトコマンドの実行結果(ステータスとリプライデータ)が送信されます。"job_once" MGCP コマンドの実行時にもこのイベントが発生します。

"job_exec" イベント送信を有効にする為に "job_config" の "mode" パラメータを 2 に設定してください。

event topic

/event/_ALL_/<node_name_from>/<application_name>/job_exec

<node_name_from> には MGCP デバイスにユーザーが設定したノード名文字列が入ります。

<application_name> は MGCP デバイスで使用しているファームウエア種別を示す文字列が入ります。

例：/event/_ALL_/STICK001/MGCP_M5Stick_Pro/job_exec

event data

{"data":"<hexstr>"}

<hexstr> には実行したダイレクトコマンドの結果ステータスとリプライデータが 16進数文字列で入ります。

ダイレクトコマンドの詳細は "ダイレクトコマンド" の章を参照して下さい。

例：{"data":"010001020304"}

8 MGCPコマンド(リモートコマンド)

マニュアル中の request 部分の記述について

● リクエスト文字列全体は JSON フォーマットに従って記述します。MGCP デバイスから送信するリプライ文字列やイベント文字列も JSON フォーマットに則っています。

● JSON 中のキー名は必ずダブルコートで囲みます。例："key":"value"

● JSON 中のキーに対応する値が文字列や16進数文字列の場合にはダブルコートで囲みます。例："key":"value"

● JSON 中のキーに対応する値が数値の場合にはダブルコートで囲まずにそのまま記述します。例："key":1234

● JSON 中のキーに対応する値が数値や文字列を格納した配列の場合には、そのぞれの数値や文字列を JSON 配列形式で指定します。例："key":[1234,"value","0FFE",5678]

● "gpio":<gpio# 0-39> の記述の場合には "gpio":11 の様に 0 から 39 までの数値を指定します。

● "level":<0|1> の記述の場合には "level":0 の様に 0 または 1 の数値を指定します。

● "mode":<"off"|"disable"|"in"|"out"|"in_out"|"out_od"|"in_out_od"> の記述の場合には "mode":"in" の様に 選択肢のなかから１つの文字列をダブルコートで囲んで指定します。

● "addr":"<hexstr>"の記述の場合には "addr":"0F" の様に 16進数表示形式の文字列をダブルコートで囲んで指定します。この時必ず１バイトあたり 2 文字を指定します。

● "value":<number|"<string>"|"<hexstr>"> の記述の場合には数値、ダブルコートで囲んだ任意の文字列、またはダブルコートで囲んだ16進数表示形式の文字列の何れか１つを指定します。

● "data":["<hexstr#1>","<hexstr#2>", .. ,"<hexstr#n>"] の記述の場合には、複数の 16進数表示形式の文字列を並べて鍵括弧で囲んだ JSON配列で指定します。例："data":["0F","AA0F","0C"]

● *[,"value":<0|1>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパラメータや文字列は省略可能です。

8.1 version

ノード名、プログラム名、バージョン番号、ライセンス情報を取得する。

request

{"command":"version"}

success reply

{"result":"success","program":"<program_name>","version":"<version_string>"
,"license":"<license_message>","nodename":"<node_name>"}

8.2 reset

CPU をリセットしてMGCP プログラムを再起動させる。

コマンド実行直後にプロセッサがリセットされます。その後、コンフィギュレーション・コンソールの "save" コマンドまたは MGCPコマンドの "save" コマンドによって、最後に不揮発領域に保存した設定値をロードして MGCPデバイスが再起動します。

リセット後に MGCP デバイスは Wi-Fi に再接続を試みます。Wi-Fi接続確立後に DHCP サーバーによってアサインされる IP アドレスは、リセット前の IPアドレスとは異なっている場合があります。

このコマンドを実行すると接続中の MQTT ブローカとの socket 接続は中断されて、リセット後に再接続を試みます。MQTT ブローカ再接続後に送信する MQTT-CONNECT メッセージ中の ClientID は、コンフィギュレーションで設定しているものを使用しますので、ClientID を明示的に変更していない限り前回と同じ ClientID でブローカに接続を試みます。MQTT ブローカによっては、重複した ClientID を持つ別のノードが接続を試みているとしてエラーになる場合があります。この場合は時間を置いてからデバイスを再リセットするか、別の ClientID に変更してください。

このコマンドはリプライメッセージを返しません。

request

{"command":"reset"}

reply

N/A

8.3 force_config

CPU をリセットして、MGCP プログラムを強制的にコンフィギュレーション・コンソールモードで起動します。

リセット時の MGCP デバイスの動作は "reset" コマンドと同じです。このコマンドはリプライメッセージを返しま

せん。

コマンド実行直後にデバイスがリセットされてコンソールモードになりますが、シリアルポートに接続するターミナルソフトによっては接続時にデバイスを再びリセットしてしまう場合があります。この場合に MGCP プログラムは再び通常モードで起動されてしまいます。これを防ぐには、予め通常モードで運用中のMGCP デバイスのシリアルポート(ログメッセージが出力されています)にターミナルソフトで接続した後、リモートから "force_config" コマンドを実行することで確実にコンフィギュレーション・コンソールに接続できます。

request

{"command":"force_config"}

reply

N/A

8.4 save

現在 MGCP デバイスに設定されているコンフィギュレーション設定値を不揮発領域 NVS に保存する。

コンフィギュレーション・コンソールの save コマンドと同等の動作を行う。

request

{"command":"save"}

success reply

{"result":"success"}

8.5 gpio_get

指定した gpio ポートの値を取得する。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。GPIO ポート値が High の場合には 1、Low の場合には 0 が "level" タグにセットされる。

有効な GPIO モードは"in", "in_out", "in_out_od" で、これら以外のモードはエラーになる。

request

{"command":"gpio_get","gpio":<gpio# 0-39>}

success reply

{"result":"success","error_text":"","level":<0|1>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.6 gpio_set

出力モードの GPIOポートを High または Low にする。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。GPIO ポートに High を設定する場合には "level" タグに 1を指定する。Low の場合には 0 を指定する。

有効な GPIO モードは"out", "in_out","out_od", "in_out_od" で、これら以外のモードはエラーになる。

request

{"command":"gpio_set","gpio":<gpio# 0-39>,"level":<0|1>}

success reply

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.7 gpio_mode

指定した gpio ポートの入出力モードを設定、または現在の設定値を取得する。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。

"mode" タグには設定したい入出力モードを指定する。指定可能なモードは下記のいずれか。

"off" "disable" "in" "out" "in_out" "out_od" "in_out_od"

"mode" タグを省略すると、現在設定されている入出力モードを取得する。

このコマンドで設定した入出力モードは直後に有効になります。

入出力モードを変更するときに、GPIO 初期値設定 "gpio_init" は無効("off")に変更されます。初期値を変更する必要がある場合には、コンフィギュレーション・コンソールを使用して設定して下さい。。

このコマンドで変更したコンフィギュレーションを不揮発領域に保存する場合にはコンフィギュレーション・コンソールまたは mgcp コマンドの "save" コマンドを実行してください。

query request

{"command":"gpio_mode","gpio":<gpio# 0-39>}

set request

{"command":"gpio_mode","gpio":<gpio#0-39>
,"mode":<"off"|"disable"|"in"|"out"|"in_out"|"out_od"|"in_out_od">}

success reply (query request)

{"result":"success","error_text":"","mode":<"off"|"disable"|"in"|"out"|"in_out"|"out_od"|"in_out_od">}

success reply (set request)

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.8 gpio_pull

指定した gpio ポートのプルアップ・プルダウンの設定、または現在の設定値を取得する。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。

"pull" タグには設定したいプルアップ・プルダウンを指定する。指定可能なモードは下記のいずれか。
"off" "floating" "up" "down" "up_down"

"pull" タグを省略すると、現在設定されているプルアップ・プルダウン設定値を取得する。

このコマンドで設定したプルアップ・プルダウン設定値は直後に有効になります。

プルアップ・プルダウン設定値を変更するときに、GPIO 初期値設定 "gpio_init" は無効("off")に変更されます。
初期値を変更する必要がある場合には、コンフィギュレーション・コンソールを使用して設定して下さい。。

ここで実行したコンフィギュレーションを不揮発領域に保存する場合にはコンフィギュレーション・コンソールまたは "save" MGCPコマンドで保存してください。

query request

{"command":"gpio_pull","gpio":<gpio# 0-39>}

set request

{"command":"gpio_pull","gpio":<gpio# 0-39>,"pull":<"off"|"floating"|"up"|"down"|"up_down">}

success reply (query request)

{"result":"success","error_text":"","pull":<"off"|"floating"|"up"|"down"|"up_down">}

success reply (set request)

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.9 gpio_get_all

全ての GPIO ポート値、ChangeDetect設定値、Blink設定値を取得する。

GPIO ポート値が High の場合にはGPIO番号のビット位置に 1 がセットされ、Low の場合には 0 がセットされる。全ての GPIO ポート値をまとめた 16進数文字列が返される。

ChangeDetect設定値、Blink設定値 についても同様に "on" の場合には該当するビット位置が 1 にセットされ、"off" の場合には 0 にセットされた16進数文字列が返される。

このコマンドはポート毎に設定されている入出力モードとは関係なく、強制的に現在のGPIOポート値を取得する。

request

{"command":"gpio_get_all"}

success reply

{"result":"success","error_text":"","gpio_val":"<gpio_val_hexstr>",
"change_detect":"<change_detect_hexstr>","gpio_blink":"<gpio_blink_hexstr>"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.10 gpio_blink

GPIOポートを High - Low 繰り返し出力状態にする。

繰り返しの間隔は約 700ms で、LED のブリンク表示やブザー出力に利用できます。

繰り返し指定の GPIO ポートは予め出力モードにしてください。出力モードになっていない場合は繰り返し出力は行われません。有効な GPIO モードは"out", "in_out","out_od", "in_out_od" です。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。

繰り返し対象にする場合には "flag" タグに "on" を指定する。"off" を指定すると繰り返しを停止する。

"flag" を "off" に設定した場合には、その時の GPIO 出力状態のままになります。強制的に Lowや High 状態にしたい場合には "level" パラメータを追加指定します。"level" パラメータは "flag":"off" を指定したときのみ有効です。

"flag" タグを省略すると、現在設定されている繰り返し設定値を取得する。

set request

{"command":"gpio_blink","gpio":<gpio# 0-39>,"flag":<"on"|"off"> [,"level":<0|1>]}

query request

{"command":"gpio_blink","gpio":<gpio# 0-39>}

success reply (set request)

{"result":"success","error_text":""}

success reply (query request)

{"result":"success","error_text":"","flag":<"on"|"off">}

error reply

{"result":"fail","error_text":"<error_message>"}

8.11 gpio_change_detect

GPIOポートの変化監視フラグの設定、または現在の設定値を取得する。

監視対象とする GPIO ポート番号は "gpio" タグに指定する。

変化監視対象にする場合には "flag" タグに "on" を指定する。"off" を指定すると監視対象から外す。

"flag" タグを省略すると、現在設定されている監視対象設定値を取得する。

set request

{"command":"gpio_change_detect","gpio":<gpio# 0-39>,"flag":<"on"|"off">}

query request

{"command":"gpio_change_detect","gpio":<gpio# 0-39>}

success reply (set request)

{"result":"success","error_text":""}

success reply (query request)

{"result":"success","error_text":"","flag":<"on"|"off">}

error reply

{"result":"fail","error_text":"<error_message>"}

8.12 gpio_pulse

指定した gpio ポートにパルス信号を出力する。

操作対象とする GPIO ポート番号は "gpio" タグに指定する。

パルス信号は、現在の GPIOポートの値を反転させた値で出力されます。パルス信号出力後には、GPIO値は元の値に戻ります。現在の GPIO ポート値が High の場合には Low パルス(High -> Low -> High) が出力され、Low の場合には High パルス(Low -> High -> Low) が出力されます。

パルス幅は width タグに1 から 65535 までの整数をミリセカンド単位で指定する。

有効な GPIO モードは "in_out", "in_out_od" で、これら以外のモードはエラーになる。

request

{"command":"gpio_pulse","gpio":<gpio# 0-39>,"width":<ms>}

success reply

{"result":"success","error_text":"","level":<0|1>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.13 i2c_config

I2C 機能のパラメータを設定する。"port"以外のパラメータを１つも指定しない場合は現在の設定値を返す。

コンフィギュレーションコマンド "i2c_xxxx" で設定する内容と同じですので、コンフィギュレーションコマンド項の説明も参照して下さい。

"port" は I2C ポート番号を指定する。0-1 までの合計2ポートを同時に使用できる。

"use" は I2Cポートを使用するかどうかを指定する。1:使用する, 0:使用しない。

"sda_gpio" は SDA ラインに使用する 0 から39までの GPIO ポート番号を指定する。

"scl_gpio" は SCL ラインに使用する 0 から39までの GPIO ポート番号を指定する。

"sda_pullup" は SDAラインで使用する GPIOポートの内蔵プルアップを有効にするかどうかを指定する。1:有効にする, 0:有効にしない。

"scl_pullup" は SCLラインで使用する GPIOポートの内蔵プルアップを有効にするかどうかを指定する。1:有効にする, 0:有効にしない。

"clock" は I2Cポートの周波数(Hz) を指定する。

"port" と共に他のパラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"i2c_config","port":<port# 0-1>,
[,"use":<0|1>]
[,"sda_gpio":<gpio# 0-39>]
[,"scl_gpio":<gpio# 0-39>]
[,"sda_pullup":<0|1>]
[,"scl_pullup":<0|1>]
[,"clock":<Hz 1-1000000>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","port":<port# 0-1>,
,"use":<0|1>
,"sda_gpio":<gpio# 0-39>
,"scl_gpio":<gpio# 0-39>
,"sda_pullup":<0|1>
,"scl_pullup":<0|1>
,"clock":<Hz 1-1000000> }

error reply

{"result":"fail","error_text":"<error_message>"}

8.14 i2c_write

I2Cマスターデバイスとして動作して、指定したI2Cスレーブデバイスにデータを書き込む。書き込み後に同一スレーブデバイスからデータの読み込みも可能。

予めコンフィギュレーションモードで、使用するI2Cポート番号を有効にして SDA,SCL ラインへのGPIOポートの割り当てと、プルアップ設定、クロックスピードを設定することでこのコマンドが使用できるようになります。

"port" パラメータに 0 または 1 の I2Cポート番号を指定する

"addr" パラメータにはスレーブデバイスアドレスを16進数文字列で指定する（例 "50","48","0A","A"）

"data" パラメータには書き込むデータ列を16進数文字列で指定する。（例 "1122AABB","4811", "0A0B"）
このとき、1バイトにつき 2 桁の16進数文字列を指定すること。(0xa,0xb,0xff の3バイトを書き込む場合には "0A0BFF" を指定します。"A0BFF"や"ABFF"を指定しないでください)

"read_len" パラメータに整数値を指定すると、データ書き込み後に指定したバイト数をスレーブデバイスから読み込む。読み込んだデータはリプライデータ中の "data" タグに 16進数文字列で格納される。

"delay" パラメータを指定すると、スレーブデバイス書き込み後に "read_len" パラメータでデータ読み込みするまでにウェイトを挿入します。ミリセカンド(ms) の単位で整数を指定する。パラメータ省略時や 0 を指定した場合はウェイトしない。

set request

{"command":"i2c_write","port":<port# 0-1>,"addr":"<hexstr>","data":"<hexstr>"}

query request

{"command":"i2c_write","port":<port# 0-1>,"addr":"<hexstr>","data":"<hexstr>"
,"read_len":<bytes> [,"delay":<msec>]}

success reply (set request)

{"result":"success","error_text":""}

success reply (query request)

{"result":"success","error_text":"","data":"<hexstr>"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.15 i2c_read

I2Cマスターデバイスとして動作して、指定したI2Cスレーブデバイスからデータを読み込む。

予めコンフィギュレーションモードで、使用するI2Cポート番号を有効にして SDA,SCL ラインへのGPIOポートの割り当てと、プルアップ設定、クロックスピードを設定することでこのコマンドが使用できるようになります。

"port" パラメータに 0 または 1 の I2Cポート番号を指定する

"addr" パラメータにはスレーブデバイスアドレスを16進数文字列で指定する (例 "50","48","0A","A")

"read_len" パラメータにスレーブデバイスから読み込むバイト数を指定する。読み込んだデータはリプライデータ中の "data" タグに 16進数文字列で格納される。

query request

{"command":"i2c_read","port":<port# 0-1>,"addr":"<hexstr>","read_len":<bytes>}

success reply (query request)

{"result":"success","error_text":"","data":"<hexstr>"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.16 i2c_write_block

I2Cマスターデバイスとして動作して、指定したI2Cスレーブデバイスにデータを書き込む。書き込みデータは複数のデータブロックを指定して、各ブロック毎に書き込み動作を行う。

"data" 配列パラメータ中の各データ項目毎に、I2C 書き込みトランザクション(start-write-stop) を繰り返す。

予めコンフィギュレーションモードで、使用するI2Cポート番号を有効にして SDA,SCL ラインへのGPIOポートの割り当てと、プルアップ設定、クロックスピードを設定することでこのコマンドが使用できるようになります。

"port" パラメータに 0 または 1 の I2Cポート番号を指定する

"addr" パラメータにはスレーブデバイスアドレスを16進数文字列で指定する (例 "50","48","0A","A")

"data" パラメータには書き込むデータブロックを16進数文字列の配列で指定する。
(例 ["00","AA"]、["1122AABB","4811","0A0B"])このとき、文字列には1バイトにつき 2 桁の16進数文字列を指定

する。(0xa,0xb,0xff の3バイトを書き込む場合には "0A0BFF" を指定します。"A0BFF"や"ABFF"を指定しないでください) データブロック配列には必ず 1 つ以上の16進数文字列を格納すること。

"delay" パラメータを指定すると、各データブロックの書き込み間にウェイトを挿入することができます。
ミリセカンド(ms) の単位で整数を指定する。パラメータ省略時や 0 を指定した場合はウェイトしない。

set request

{"command":"i2c_write_block","port":<port# 0-1>,"addr":"<hexstr>"
,"data":["<hexstr#1>","<hexstr#2>", .. ,"<hexstr#n>"] [,"delay":<msec>]}

注意：data タグの値に指定する配列 "data":["xxxx","xxxx", ... ,"xxxx"] は必須パラメータです。

success reply (set request)

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.17 spi_config

SPI 機能のパラメータを設定する。"port"以外のパラメータを１つも指定しない場合は現在の設定値を返す。

コンフィギュレーションコマンド "spi_xxxx" で設定する内容と同じですので、コンフィギュレーションコマンド項の説明も参照して下さい。

"port" は SPI ポート番号を指定する。0-1 までの合計2ポートを同時に使用できる。

"use" は SPIポートを使用するかどうかを指定する。1:使用する, 0:使用しない。

"miso_gpio" は SPIポートの MISO ラインに使用する 0 から39までの GPIO ポート番号を指定する。-1 はこのラインを使用しないことを示す。

"mosi_gpio" は SPIポートの MOSI ラインに使用する 0 から39までの GPIO ポート番号を指定する。-1 はこのラインを使用しないことを示す。

"clk_gpio" は SPIポートの CLK ラインに使用する 0 から39までの GPIO ポート番号を指定する。-1 はこのラインを使用しないことを示す。

"cs_gpio" は SPIポートの CS ラインに使用する 0 から39までの GPIO ポート番号を指定する。-1 はこのラインを使用しないことを示す。

"mode" は SPIポートの動作モードを指定する。0 から 3までの値が有効。

"clock" は SPIポートのクロックスピード周波数(Hz) を指定する。

"port" と共に他のパラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"spi_config","port":<port# 0-1>,
[,"use":<0|1>]
[,"miso_gpio":<gpio# 0-39 or -1>]
[,"mosi_gpio":<gpio# 0-39 or -1>]
[,"clk_gpio":<gpio# 0-39 or -1>]
[,"cs_gpio":<gpio# 0-39 or -1>]
[,"mode":<spi_mode# 0-3>]
[,"clock":<Hz 1-100000000>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","port":<port# 0-1>,
,"use":<0|1>
,"miso_gpio":<gpio# 0-39 or -1>
,"mosi_gpio":<gpio# 0-39 or -1>
,"clk_gpio":<gpio# 0-39 or -1>
,"cs_gpio":<gpio# 0-39 or -1>
,"mode":<spi_mode# 0-3>
,"clock":<Hz 1-100000000>] }

error reply

{"result":"fail","error_text":"<error_message>"}

8.18 spi_write

SPIマスターデバイスとして動作して、SPIスレーブデバイスにデータを書き込む。書き込みと同時にスレーブデバイスからデータの読み込みを行う。

予めコンフィギュレーションモードで、使用するSPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を有効にして MISO,MOSI,CLK,CS ラインへのGPIOポートの割り当てと、SPI-MODE, クロックスピードを設定することでこのコマンドを使用できます。

"port" パラメータに 0 または 1 の SPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を指定する

"data" パラメータには書き込むデータ列を16進数文字列で指定する。
(例 "1122AABB","4811", "0A0B")このとき、1バイトにつき 2 桁の16進数文字列を指定すること。
(0xa,0xb,0xff の3バイトを書き込む場合には "0A0BFF" を指定します。"A0BFF"や"ABFF"を指定しないでください)
"data" パラメータに指定可能なバイト数は 1500 まで。

"read_flag":1 を指定した場合には、"data"パラメータに指定したバイト数と同じバイト数をスレーブデバイスから読み込む。読み込んだデータはリプライデータ中の "data" タグに 16進数文字列で格納される。"read_flag":0 を指定した場合や "read_flag" パラメータ自身を省略した場合には、スレーブデバイスへの書き込みのみを行い "data"タグは返らない。

set request

{"command":"spi_write","port":<port# 0-1>,"data":"<hexstr>" [,"read_flag":0]}

query request

{"command":"spi_write","port":<port# 0-1>,"data":"<hexstr>","read_flag":1}

success reply (set request)

{"result":"success"}

success reply (query request)

{"result":"success","data":"<hexstr>"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.19 spi_read

SPIマスターデバイスとして動作して、SPIスレーブデバイスにからデータを読み込む。

予めコンフィギュレーションモードで、使用するSPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を有効にして MISO,CLK,CS ラインへのGPIOポートの割り当てと、SPI-MODE, クロックスピードを設定することでこのコマンドを使用できます。

"port" パラメータに 0 または 1 の SPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を指定する

"read_len" パラメータにスレーブデバイスから読み込むバイト数を指定する。指定可能な最大バイト数は 1500。
読み込んだデータはリプライデータ中の "data" タグに 16進数文字列で格納される。

query request

{"command":"spi_read","port":<port# 0-1>,"read_len":<bytes>}

success reply (query request)

{"result":"success","data":"<hexstr>"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.20 spi_write_block

SPIマスターデバイスとして動作して、SPIスレーブデバイスにデータを書き込む。書き込みデータは複数のデータブロックを指定して、各ブロック毎に書き込み動作を行う。

"data" 配列パラメータ中の各データ項目毎に、SPI書き込みトランザクションを繰り返す。

予めコンフィギュレーションモードで、使用するSPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を有効にして
MOSI,CLK,CS ラインへのGPIOポートの割り当てと、SPI-MODE, クロックスピードを設定することでこのコマンドを使用できます。

"port" パラメータに 0 または 1 の SPIポート番号(0:HSPI_HOST,1:VSPI_HOST)を指定する

"data" パラメータには書き込むデータブロックを16進数文字列の配列で指定する。
(例 ["00","AA"]、["1122AABB","4811","0A0B"])
このとき、文字列には1バイトにつき 2 桁の16進数文字列を指定する。
(0xa,0xb,0xff の3バイトを書き込む場合には "0A0BFF" を指定します。"A0BFF"や"ABFF"を指定しないでください)
データブロック配列には必ず 1 つ以上の16進数文字列を格納すること。

"delay" パラメータを指定すると、各データブロックの書き込み間にウェイトを挿入することができます。
ミリセカンド(ms) の単位で整数を指定する。パラメータ省略時や 0 を指定した場合はウェイトしない。

"dc_gpio" パラメータを指定すると各データブロック転送前に、GPIO ポート値に 1 または 0 を出力します。
１データブロック転送毎に、GPIO ポート値を反転して出力します。スレーブデバイスの(data/command)切り替えラインとして使用することを想定しています。

"level" パラメータに最初のデータブロック転送時の GPIO ポート値を指定できます。"level" パラメータ省略時は

0 が設定されます。"dc_gpio" パラメータを省略するか、-1 を指定した場合には GPIO ポートへの出力は行いません。

set request

{"command":"spi_write_block","port":<port# 0-1> *[,"delay":<msec>] [,"dc_gpio":<gpio# 0-39>*
[,"level":<0|1>]] ,"data":["<hexstr#1>","<hexstr#2>", .. ,"<hexstr#n>"] }

注意：data タグの値に指定する配列 "data":["xxxx","xxxx", ... ,"xxxx"] は必須パラメータです。

success reply (set request)

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.21 lcd_print(M5Stack,M5Stick-C専用)

M5Stack/M5Stick-C 搭載 LCD ディスプレイに ASCII文字列を描画する。

"text" パラメータに描画する文字列を指定する。ASCII 文字のみ有効です。

"x" パラメータに X 軸(横)方向の描画開始位置を指定する。
このパラメータを省略した場合には、文字列全体を中央位置に表示できるように自動調整される。

"y" パラメータに Y 軸(縦)方向の描画開始位置を指定する。
このパラメータを省略した場合には、画面の中央に表示する位置に設定される。
デフォルトフォント("font":2) を使用する場合には M5Stick-C には最大３行表示可能で、それぞれの
描画開始位置の値は (1行目 "y":0),(2行目 "y":27),(3行目 "y":55) を指定してください。

"wrap" に 1 に指定すると長い文字列を折り返し表示します。このとき "x" パラメータと "y" パラメータ省略時のデフォルト値は共に 0 に設定されます。"wrap"指定はコマンド実行時のみ有効でデバイス側には保存されません。
このため "wrap" を有効にするためには毎回 "wrap" パラメータに 1 を指定します。
"wrap" パラメータ省略時は 0 に指定されます。

"clear" パラメータにカラーコード(0 から 29までの整数値) を指定すると、文字列描画前に指定した色で画面を塗りつぶします。カラーコードと色の対応は lcd_preset_color コマンドの項を参照してください。

"font" パラメータには表示フォントを指定できます。
フォント番号には 0 から 9 までの整数を指定します。デフォルトフォントは 2 です。
フォントを指定するとその設定値がデバイス上で保存されて、次に "font" パラメータを指定するかデバイスをリセ

ットするまで同じ設定値を使用します。

"fg" 文字列描画時のフォアグランドカラーを指定できます。カラーコードの意味は "clear" パラメータと同じです。フォアグランドカラーを指定するとその設定値がデバイス上で保存されて、次に "fg" パラメータを指定するかデバイスをリセットするまで同じ設定値を使用します。

"bg" 文字列描画時のバックグランドカラーを指定できます。カラーコードの意味は "clear" パラメータと同じです。バックグランドカラーを指定するとその設定値がデバイス上で保存されて、次に "bg" パラメータを指定するかデバイスをリセットするまで同じ設定値を使用します。

request

{"command":"lcd_print","text":<string> *[,"x":<x_pos>] [,"y":<y_pos>] [,"wrap":<1|0>]*
[,"clear":<color# 0-29>]
[,"font":<font# 0-9>] [,"fg":<color# 0-29>] [,"bg":<color# 0-29>]}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.22 lcd_config(M5Stick-C専用)

M5Stick-C 搭載 LCD ディスプレイの "向き"と"明るさ"を設定する。パラメータ省略時は現在の設定値を取得する。M5Stack 用 MGCP はこのコマンドはサポートしていません。

"rotation" パラメータには画面の "向き" を表す番号を指定する。1:LANDSCAPE, 3:LANDSCAPE_FLIP (1 または 3 のみ有効)

"backlight" パラメータには LCD バックライトの明るさを指定する。0 から 7 までの整数値を指定する。

このコマンドで設定した値は直後にLCD ディスプレイに反映します。

パラメータを一つも指定しなかった場合には現在の設定値を返します。

ここで実行したコンフィギュレーションを不揮発領域に保存する場合にはコンフィギュレーション・コンソールまたは mgcp コマンドの "save" コマンドで保存してください。

request

{"command":"lcd_config" *[,"rotation":<rot# 1|3>] [,"backlight":<power# 0-7>]*}

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","rotation":<rot# 1|3>,"backlight":<power# 0-7>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.23 lcd_draw(M5Stack,M5Stick-C専用)

M5Stack/M5Stick-C 搭載 LCD ディスプレイに図形を描画する。

１つのコマンドで複数の図形を一度に描画指定することも可能。

"clear" パラメータにカラーコード(0 から 29までの整数値) を指定すると、図形描画前に指定した色で画面を塗りつぶす。カラーコードと色の対応については lcd_preset_color コマンドを参照の事。

"circle" パラメータを指定すると円を描画する。配列には 中心座標の X 軸(横)の値と Y 軸(縦)の値,
半径, カラーコードを指定する。複数の円を指定する場合には、同じパラメータ順で値を追加指定する。

"triangle" パラメータを指定すると三角形を描画する。配列には ３つの頂点座標それぞれの X 軸(横)の値と Y 軸(縦)の値、カラーコードを指定する。複数の三角形を指定する場合には、同じパラメータ順で値を追加指定する。

"round_rect" パラメータを指定すると角丸四角形を描画する。配列には描画開始座標の X 軸(横)の値と Y 軸(縦)の値,幅、高さ、丸角の半径、カラーコードを指定する。複数の角丸四角形を指定する場合には、同じパラメータ順で値を追加指定する。

"rect" パラメータを指定すると四角形を描画する。配列には描画開始座標の X 軸(横)の値と Y 軸(縦)の値,
幅、高さ、カラーコードを指定する。複数の四角形を指定する場合には、同じパラメータ順で値を追加指定する。

"line" パラメータを指定すると直線を描画する。配列には 始点と終点の座標それぞれの X 軸(横)の値と Y 軸(縦)の値,カラーコードを指定する。複数の直線を指定する場合には、同じパラメータ順で値を追加指定する。

"pixel" パラメータを指定すると点を描画する。配列には 座標の X 軸(横)の値と Y 軸(縦)の値,
カラーコードを指定する。複数の点を指定する場合には、同じパラメータ順で値を追加指定する。

"fill" に 1 に指定すると図形描画時に指定したカラーコードの色で図形内部を塗りつぶす。
パラメータ省略時は 0 が指定される。複数の図形を描画している場合には全ての図形が塗りつぶし対象となる。図

形を個別に塗りつぶし指定する場合には lcd_draw を複数回コールすること。

request

{"command":"lcd_draw" [,"clear":<color# 0-29>]
[,"circle":[x0,y0,r0,c0[,x1,y1,r1,c1 .. ,xN,yN,rN,cN]]]
[,"triangle":[xa0,ya0,xb0,yb0,xc0,yc0,c0[,xa1,ya1,xb1,yb1,xc1,yc1,c1 .. ,xaN,yaN,xbN,ybN,xcN,ycN,cN]]]
[,"round_rect":[x0,y0,w0,h0,r0,c0[,x1,y1,w1,h1,r1,c1 .. ,xN,yN,wN,hN,rN,cN]]]
[,"rect":[x0,y0,w0,h0,c0[,x1,y1,w1,h1,c1 .. ,xN,yN,wN,hN,cN]]]
[,"line":[xs0,ys0,xe0,ye0,c0[,xs1,ys1,xe1,ye1,c1 .. ,xsN,ysN,xeN,yeN,cN]]]
[,"pixel":[x0,y0,c0[,x1,y1,c1 .. ,xN,yN,cN]]]
[,"fill":<0|1>] }

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.24 lcd_bitmap(M5Stack,M5Stick-C専用)

M5Stack/M5Stick-C 搭載 LCD ディスプレイにビットマップを描画する。

１つのコマンドで複数のビットマップを一度に描画指定することも可能。

座標 (x#n,y#n) にビットマップを描画する。ビットマップは左上を描画開始座標に合わせる。
"data" パラメータ <hexstr#n> 中の各ビットマップデータが bit値 1 の場合には "fg" パラメータで指定されたカラーコードを使用して、bit値 0 の場合には "bg" パラメータで指定されたカラーコードで LCD にピクセルを描画する。

"data" パラメータに指定する <hexstr#n> ビットマップデータの構成は下記の様になります。

ビットマップデータフォーマット
例: 幅(w): 12ピクセル, 高さ(h):3ピクセル の場合

data パラメータに指定するデータ長は 6 バイトで下記の並びに配置されたデータが格納されます。

buff[] length = 6 bytes
(x0,y0)
    +---
w ->| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|

```
   +----------------------+----------------------+
   |b7|b6|b5|b4|b3|b2|b1|b0|b7|b6|b5|b4|xx|xx|xx|xx| xx = don't care
 h +----------------------+----------------------+
 0 |         buff[0]       |       buff[1]         |
 1 |         buff[2]       |       buff[3]         |
 2 |         buff[4]       |       buff[5]         |
```

"clear" パラメータにカラーコード(0 から 29までの整数値) を指定すると、図形描画前に指定した色で画面を塗りつぶす。カラーコードと色の対応については lcd_preset_color コマンドを参照の事。

"fg" ビット値 1 のピクセル描画時のカラーを指定できます。
フォアグランドカラーを指定するとその設定値がデバイス上で保存されて、次に "fg" パラメータを指定するかデバイスをリセットするまで同じ設定値を使用します。

"bg" ビット値 0 のピクセル描画時のカラーを指定できます。
バックグランドカラーを指定するとその設定値がデバイス上で保存されて、次に "bg" パラメータを指定するかデバイスをリセットするまで同じ設定値を使用します。

"size" パラメータに 1 以上の整数を指定すると、ビットマップを size 倍に拡大して描画する。パラメータ省略時は 1が指定される。

"data"パラメータには描画するビットマップデータの開始座標(x#n,y#n)と横縦のピクセルサイズ(w#n,h#n)、ビットマップデータを16進数文字列を配列に格納して指定する。複数のビットマップを指定する場合には、同じパラメータ順で値を追加指定する。

request

```
{"command":"lcd_bitmap"
      [,"clear":<color# 0-29>]
      [,"fg":<color# 0-29>]
      [,"bg":<color# 0-29>]
      [,"size":<magnify# 1-10>]
      ,"data":[x0,y0,w0,h0,"<hexstr0>"[,x1,y1,w1,h1,"<hexstr1>" .. ,xN,yN,wN,hN,"<hexstrN>"]]}
```

success reply

```
{"result":"success"}
```

error reply

```
{"result":"fail","error_text":"<error_message>"}
```

8.25 lcd_bitmap2(M5Stack,M5Stick-C専用)

M5Stack/M5Stick-C 搭載 LCD ディスプレイにビットマップを描画する。lcd_bitmap コマンドとは違ってビット値 1 を指定された部分のみを描画する。

１つのコマンドで複数のビットマップとカラーを指定して一度に描画指定することも可能。

座標 (x#n,y#n) にビットマップを描画する。ビットマップは左上を描画開始座標に合わせる。

"data" パラメータ <hexstr#n> 中の各ビットマップデータが bit値 1 の場合のみ "c#n" パラメータで指定されたカラーコードを使用してピクセルを描画する。bit 値が 0 の場合にはそのピクセルは現在の色を維持する。

"data" パラメータに指定する <hexstr#n> ビットマップデータの構成は lcd_bitmap コマンドと同様ですので、lcd_bitmap コマンドの説明を参照してください。

"clear" パラメータにカラーコード(0 から 29までの整数値) を指定すると、図形描画前に指定した色で画面を塗りつぶす。カラーコードと色の対応については lcd_preset_color コマンドを参照の事。

"size" パラメータに 1 以上の整数を指定すると、ビットマップを size 倍に拡大して描画する。パラメータ省略時は 1が指定される。

"data"パラメータには描画するビットマップデータの開始座標(x#n,y#n)と横縦のピクセルサイズ(w#n,h#n)、カラーコード(c#n)、ビットマップデータを16進数文字列を配列に格納して指定する。複数のビットマップを指定する場合には、同じパラメータ順で値を追加指定する。

request

{"command":"lcd_bitmap2"

[,"clear":<color# 0-29>*]*

[,"size":<magnify# 1-10>*]*

,"data":[x0,y0,w0,h0,c0,"<hexstr0>"*[*,x1,y1,w1,h1,c1,"<hexstr1>" .. ,xN,yN,wN,hN,cN,"<hexstrN>"*]*] }

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.26 lcd_preset_color(M5Stack,M5Stick-C専用)

M5Stack/M5Stick-C 搭載 LCD ディスプレイ描画時のカラーパレットを設定する

"rgb" パラメータを省略すると現在のカラーパレット(RGB値)を取得する。

"color" パラメータで指定したカラーコード(0 から 29までの整数値) の色を、"rgb" パラメータに指定した RGB のそれぞれの色成分で設定する。

カラーパレットは描画コマンド実行時にのみ使用して、LCD描画後にカラーコードを変更してもピクセルの色に影響を与えません。

デバイス起動直後は、0 から 18 までのカラーコードは下記の色にプリセットされます。
19 から 29 までのカラーコードはユーザー用のエリアですが、0 から 18 部分もこのコマンドで変更可能です。

このコマンドで設定したカラーパレットはデバイスをリセットすると破棄されます。

[color code]	preset color	
[0]	BLACK	
[1]	NAVY	
[2]	DARKGREEN	
[3]	DARKCYAN	
[4]	MAROON	
[5]	PURPLE	
[6]	OLIVE	
[7]	LIGHTGREY	
[8]	DARKGREY	
[9]	BLUE	
[10]	GREEN	
[11]	CYAN	
[12]	RED	
[13]	MAGENTA	
[14]	YELLOW	
[15]	WHITE	
[16]	ORANGE	
[17]	GREENYELLOW	
[18]	PINK	
[19..29]	WHITE	ユーザーが自由に使用可能

request

{"command":"lcd_preset_color","color":<color# 0-29> [,"rgb":[<r# 0..255>,<g# 0..255>,<b# 0..255>]] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","rgb":[<r# 0..255>,<g# 0..255>,<b# 0..255>]}

error reply

{"result":"fail","error_text":"<error_message>"}

8.27 neopixel_config

NeoPixel デバイスの各種パラメータを設定する。パラメータを１つも指定しない場合は現在の設定値を返す。
コンフィギュレーションコマンド "neopixel_xxxx" で設定する内容と同じですので、コンフィギュレーションコマンド項の説明も参照して下さい。

"use" は NeoPixel デバイスを使用するかどうかを指定する。1:使用する, 0:使用しない。

"data_gpio" は NeoPixelデバイスを接続する GPIO ポート番号を指定する。

"count" は NeoPixel デバイスに接続されている LED 個数を指定する。

"brightness" は LED 全体の明るさを 0-255 のスケールで指定する。

"blink_fade" は "neopixel_blink" コマンドでブリンク表示するときのフェード時間の(1/10)をミリ秒で指定する。

"blink_width" は "neopixel_blink" コマンドでブリンク表示するときの点灯と消灯時間をミリ秒で指定する。

"maxmilliamps" は NeoPixel デバイスで消費する最大電流値を指定する。単位は mA で 0 で無制限になる。
この指定は明るさをソフト的にコントロールしていますので、厳密な制御は行いません。

"mode" は 下記の NeoPixel 表示モードを指定します。
MGCPコマンド "neopixel_draw", "neopixel_blink" を実行すると、自動的に "mode" が 0 に設定されます。

0:通常モード(MGCP デバイス起動時のデフォルト)
1:テストパターン表示
2:現在表示中の LED パターンをシフト表示する(現在未使用)
3:ダイレクト表示(現在未使用)

"use", "data_gpio", "count" パラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハー

ドリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"neopixel_config" [,"use":<0|1>]
[,"data_gpio":<gpio# 0|2|4|5|12|13|14|15|16|17|18|19|21|22|23|25|26|27>]
[,"count":<LEDs 1-3000>]
[,"brightness":<0-255>]
[,"blink_fade":<1-65535>]
[,"blink_width":<1-65535>]
[,"maxmilliamps":<mA 0-65535>]
[,"mode":<disp_mode 0-3>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","use":<0|1>,"data_gpio":<gpio# 0|2|4|5|12|13|14|15|16|17|18|19|21|22|23|25|26|27>,
"count":<LEDs 1-3000>,"brightness":<0-255>,"blink_fade":<1-65535>,"blink_width":<1-65535>,
"maxmilliamps":<mA 0-65535>,"mode":<disp_mode 0-3>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.28 neopixel_draw

NeoPixel デバイスにピクセルを描画する。

コンフィギュレーションで NeoPixel を有効にしている場合のみ実行可能。

"clear" パラメータにカラーコード(0 から 199までの整数値) を指定すると、NeoPixel デバイスに搭載された全LEDを指定した色で点灯させる。カラーコードと色の対応については neopixel_preset_color コマンドを参照の事。

"pixel" パラメータは指定した色で LED を点灯させる。配列には LED 位置(0 からLED全個数-1までの整数)とカラーコードを指定する。複数の点を指定する場合には、LED位置とカラーコードの順でパラメータを追加できる。パラメータで指定しなかった位置の LED は変化しない。

"pixel_rgb" パラメータを指定すると指定した LED を点灯させる。配列には LED 位置(i)と表示色の RGB 値を指定する。複数の点を指定する場合には、同じパラメータ順で値を追加指定する。RGB 値は 0 .. 255 の整数値を指定する。パラメータで指定しなかった位置の LED は変化しない。

request

{"command":"neopixel_draw" [,"clear":<color# 0-199>]

[,"pixel":[i0,c0[,i1,c1 .. ,iN,cN]]]

[,"pixel_rgb":[i0,r0,g0,b0[,i1,r1,g1,b1 .. ,iN,rN,gN,bN]]]

}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.29 neopixel_blink

NeoPixel LEDを点滅表示または点灯表示状態にする。

コンフィギュレーションで NeoPixel を有効にしている場合のみ実行可能。

"on" パラメータに "all" を指定すると全ての LED を点滅させる。配列形式で複数の LED 位置(0 からLED全個数-1までの整数)を指定すると、そのLED のみを点滅表示にする。パラメータで指定しなかった位置の LED 点滅状態は変化しない。

"off" パラメータに "all" を指定すると全ての LED を通常表示にする。配列形式で複数の LED 位置(0 からLED全個数-1までの整数)を指定すると、そのLED のみを通常表示にする。パラメータで指定しなかった位置の LED 点滅状態は変化しない。

LED 点滅の間隔やフェードの設定は "neopixel_config" コマンドで指定できる。

request

{"command":"neopixel_blink" [,"on":"all"]

[,"on":[i0,i1, .. ,iN]]

[,"off":"all"]

[,"off":[i0,i1, .. ,iN]]

}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.30 neopixel_preset_color

NeoPixel LED描画時のカラーパレットを設定する。

コンフィギュレーションで NeoPixel を有効にしている場合のみ実行可能。

"rgb" パラメータを省略すると現在のカラーパレット(RGB値)を取得する。

"color" パラメータで指定したカラーコード(0 から 199までの整数値) の色を、"rgb" パラメータに指定した RGB のそれぞれの色成分で設定する。

カラーパレットは描画コマンド実行時にのみ使用して、LCD描画後にカラーコードを変更してもピクセルの色に影響を与えません。

デバイス起動直後は、0 から 147 までのカラーコードは下記の色にプリセットされます。148 から 199 までのカラーコードはユーザー用のエリアですが、0 から 147 部分もこのコマンドで変更可能です。

このコマンドで設定したカラーパレットはデバイスをリセットすると破棄されます。

[color code]	preset color
0	AliceBlue
1	Amethyst
2	AntiqueWhite
3	Aqua
4	Aquamarine
5	Azure
6	Beige
7	Bisque
8	Black
9	BlanchedAlmond
10	Blue
11	BlueViolet
12	Brown
13	BurlyWood
14	CadetBlue
15	Chartreuse
16	Chocolate
17	Coral
18	CornflowerBlue
19	Cornsilk

20	Crimson
21	Cyan
22	DarkBlue
23	DarkCyan
24	DarkGoldenrod
25	DarkGray
26	DarkGrey
27	DarkGreen
28	DarkKhaki
29	DarkMagenta
30	DarkOliveGreen
31	DarkOrange
32	DarkOrchid
33	DarkRed
34	DarkSalmon
35	DarkSeaGreen
36	DarkSlateBlue
37	DarkSlateGray
38	DarkSlateGrey
39	DarkTurquoise
40	DarkViolet
41	DeepPink
42	DeepSkyBlue
43	DimGray
44	DimGrey
45	DodgerBlue
46	FireBrick
47	FloralWhite
48	ForestGreen
49	Fuchsia
50	Gainsboro
51	GhostWhite
52	Gold
53	Goldenrod
54	Gray
55	Grey
56	Green
57	GreenYellow
58	Honeydew

59	HotPink
60	IndianRed
61	Indigo
62	Ivory
63	Khaki
64	Lavender
65	LavenderBlush
66	LawnGreen
67	LemonChiffon
68	LightBlue
69	LightCoral
70	LightCyan
71	LightGoldenrodYellow
72	LightGreen
73	LightGrey
74	LightPink
75	LightSalmon
76	LightSeaGreen
77	LightSkyBlue
78	LightSlateGray
79	LightSlateGrey
80	LightSteelBlue
81	LightYellow
82	Lime
83	LimeGreen
84	Linen
85	Magenta
86	Maroon
87	MediumAquamarine
88	MediumBlue
89	MediumOrchid
90	MediumPurple
91	MediumSeaGreen
92	MediumSlateBlue
93	MediumSpringGreen
94	MediumTurquoise
95	MediumVioletRed
96	MidnightBlue
97	MintCream

98	MistyRose
99	Moccasin
100	NavajoWhite
101	Navy
102	OldLace
103	Olive
104	OliveDrab
105	Orange
106	OrangeRed
107	Orchid
108	PaleGoldenrod
109	PaleGreen
110	PaleTurquoise
111	PaleVioletRed
112	PapayaWhip
113	PeachPuff
114	Peru
115	Pink
116	Plaid
117	Plum
118	PowderBlue
119	Purple
120	Red
121	RosyBrown
122	RoyalBlue
123	SaddleBrown
124	Salmon
125	SandyBrown
126	SeaGreen
127	Seashell
128	Sienna
129	Silver
130	SkyBlue
131	SlateBlue
132	SlateGray
133	SlateGrey
134	Snow
135	SpringGreen
136	SteelBlue

137	Tan	
138	Teal	
139	Thistle	
140	Tomato	
141	Turquoise	
142	Violet	
143	Wheat	
144	White	
145	WhiteSmoke	
146	Yellow	
147	YellowGreen	
148..199	White	ユーザーが自由に使用可能

request

{"command":"neopixel_preset_color","color":<color# 0-199> /,"rgb":[<r# 0..255>,<g# 0..255>,<b# 0..255>]/}

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","rgb":[<r# 0..255>,<g# 0..255>,<b# 0..255>]}

error reply

{"result":"fail","error_text":"<error_message>"}

8.31 pwm_config

PWM 機能のパラメータを設定する。"ch"以外のパラメータを１つも指定しない場合は現在の設定値を返す。

コンフィギュレーションコマンド "pwm_xxxx" で設定する内容と同じですので、コンフィギュレーションコマンド項の説明も参照して下さい。

"ch" は PWM 出力用のチャンネルを指定する。0-3 までの合計４チャンネルを同時に使用できる。

"pwm_use" は PWM チャンネルを使用するかどうかを指定する。1:使用する, 0:使用しない。

"servo_use" は PWM チャンネルがサーボ用に設定されているかどうかを示す。この値が 1 の場合には SERVO デバイスに適していない PWM 波形になるのを防ぐために "pwm_config" コマンドではなく、"servo_config" MGCP コマンドを使用して詳細パラメータを設定・取得して下さい。

"gpio" は PWM 信号出力に使用する GPIO ポート番号を指定する。

"resolution" は PWM Duty 値の分解能をビット数で指定する。PWM Duty 値は 0 から (2^resolution - 1) 間の値を指定できる。例えば 16 を指定すると 0 - 65535 の Duty 値を指定できる。
"resolution" 値を設定すると強制的に現在の PWM Duty 値は 0 になります。0 以外に設定したい場合は "duty" パラメータで指定します。

"freq" は PWM 周波数(Hz) を指定する。
周波数のみを設定・参照する場合には "pwm_freq" MGCP コマンドも使用できます。

"duty" は PWM Duty 値を 0 から (2^resolution - 1) 間の値で指定する。
現在出力中の PWM Duty値を設定・参照する場合には "pwm_duty" MGCP コマンドを使用してください。

"pwm_use", "gpio", "resolution" パラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

> ⚠ **PWM 周波数と分解能にはトレードオフの関係があります**
> PWM 周波数を高くする場合には、それに合わせて分解能を下げる必要があります。
> 例えばPWM周波数 5 KHz の場合の最大分解能は 13ビットになります。設定可能な最大周波数 40MHzの場合の分解能は 1ビットになります。詳しくは ESP32 プロセッサの "ESP32 Technical Reference Manual" 中の "LED PWM Controller" をご覧ください。

request

{"command":"pwm_config","ch":<0-3>
[,"pwm_use":<0|1>]
[,"gpio":<gpio# 0-39>]
[,"resolution":<resolution# 0-20>]
[,"freq":<frequency# Hz>]
[,"duty":<0-(2^resolution - 1)>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","ch":<ch# 0-3>,"pwm_use":<0|1>,"servo_use":<0|1>,"gpio":<gpio# 0-39>,
"resolution":<0-20>,"freq":<frequency Hz>,"duty":<0-(2^resolution - 1)>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.32 pwm_freq

PWM 周波数の設定または現在の設定値の取得。

"freq" は PWM 周波数(Hz) を指定する。

SERVO 信号用に PWM 機能を使用する場合には、このコマンドで直接 SERVO(PWM)周波数を指定しないで代わりに、"servo_config" MGCPコマンドを使用してください。

request

{"command":"pwm_freq","ch":<0-3>[,"freq":<frequency# Hz>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","ch":<ch# 0-3>,"freq":<frequency Hz>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.33 pwm_duty

PWM Duty値の設定または現在の設定値の取得。

"duty" は PWM Duty 値を 0 から (2^resolution - 1) 間の値で指定する。

SERVO 信号用の SERVO 位置を設定・参照する場合には、このコマンドで直接 PWM Duty値を指定しないで代わりに、"servo_pos" MGCPコマンドを使用してください。

request

{"command":"pwm_duty","ch":<0-3>[,"duty":<0-(2^resolution - 1)>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","ch":<ch# 0-3>,"duty":<0-(2^resolution - 1)>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.34 pwm_blink

PWM 出力 のOn/Off を繰り返す。繰り返しの間隔は約 700ms で固定、On 期間時には PWM チャンネルに設定された Duty 値の波形を出力して、Off 期間には High または Low 信号を出力する。

操作対象とする PWM チャンネルを "ch" パラメータに指定する。

繰り返し対象にする場合には "flag" タグに "on" を指定する。"off" を指定すると繰り返しを停止する。停止後は最後に設定した PWM Duty値による連続出力状態になる。

"flag" タグを省略すると、現在設定されている繰り返し設定値を取得する。
"level" パラメータには PWM Off(アイドル)期間の GPIO 出力レベルを指定します。省略時は 0(Low) になります。
"duty" は PWM Duty 値を 0 から (2^resolution - 1) 間の値で指定する。省略時は PWMチャンネルに設定した最後の Duty 値を使用します。

request

{"command":"pwm_blink","ch":<0-3> *[,"flag":<"on"|"off">]* *[,"level":<0|1>]* *[,"duty":<0-(2^resolution - 1)>]*}

success reply (set request)

{"result":"success","error_text":""}

success reply (query request)

{"result":"success","error_text":"","ch":<0-3>,"flag":<"on"|"off">,
"level":<0|1>,"duty":<0-(2^resolution - 1)>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.35 pwm_stop

PWM 信号出力を停止する。

"level" パラメータには PWM アイドル時の出力レベルを指定します。省略時は 0 になります。

再び PWM信号を出力する場合には "pwm_duty" または "servo_pos" MGCPコマンドを実行してください。

request

{"command":"pwm_stop","ch":<0-3> [,"level":<0|1>]}

success reply (set request)

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.36 servo_config

PWM 機能を SERVO 信号用に設定する。"ch"以外のパラメータを１つも指定しない場合は現在の設定値を返す。

PWM 周波数や分解能、Duty値を SERVO 信号出力に適したコンフィギューレションに設定します。このコマンドで設定した PWM 設定値は "pwm_config" コマンドで確認することもできます。

"servo_config コマンドで設定した内容は、"pwm_config" コマンドで下記パラメータ指定したのと同じ処理を行う。

"ch":<0-3> ,"pwm_use":<0|1>,"gpio":<gpio# 0-39>,"resolution":16,"freq":<Hz 50-400>,"duty":<"freq"のみを指定した場合は下記SERVO中間位置の duty値を設定する。"pos" 指定時は下記の計算で求めた duty値が設定される>

"servo_config" コマンドで設定された SERVO 信号の主なサーボ位置の PWM Duty値は下記の様になります。

"freq" で指定した周波数が 50Hz (デフォルト値)の場合の例：

duty 分解能(resolution) = 16bit =(0 から 65535)
servo周波数 が "freq":50(Hz) の周期は 1/50 = 20ms なので
SERVO 最小位置(600)の PWM duty値 は (600us / 20ms) * 65535 = 1966
SERVO 中間位置(1500)の PWM duty値 は (1500us / 20ms) * 65535 = 4915
SERVO 最大位置(2400)の PWM duty値 は (2400us / 20ms) * 65535 = 7864

"ch" は PWM 出力用のチャンネルを指定する。0-3 までの合計４チャンネルを同時に使用できる。

"servo_use" は PWM チャンネルを SERVO デバイス用に使用するかどうかを指定する。1:使用する, 0:使用しない。"servo_use"を設定した場合には、PWM チャンネルを使用するかどうかを示す "pwm_use" も同値に設定します。

"gpio" は PWM 信号出力に使用する GPIO ポート番号を指定する。

"freq" は SERVO(PWM) 周波数(Hz) を指定する。周波数は 50 から 400 までを指定できる。パラメータ省略時は 50Hz になります。

"pos" は SERVO位置を指定する。単位はマイクロ秒で 500(us) から 2500(us)の範囲で指定する。パラメータ省略時は SERVO中間位置の 1500(us)になります。"servo_config"コマンドに続けて "save" MGCP コマンドを実行すると、最後に "pos" パラメータで設定した SERVO位置に該当する PWM Duty値がコンフィギュレーションに保存されます。

これによってデバイスリセット後の SERVO 初期位置を設定することができます。現在 SERVO 信号出力中で SERVO デバイスを操作する時の SERVO位置パラメータ設定には、"servo_pos" MGCP コマンドを使用してください。

"servo_use", "gpio","freq","pos" パラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"servo_config","ch":<0-3>

[,"servo_use":<0|1>]

[,"gpio":<gpio# 0-39>]

[,"freq":<Hz 50-400>]

[,"pos":<pos_us# 500-2500>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","ch":<ch# 0-3>,"servo_use":0}

または、

{"result":"success","ch":<ch# 0-3>,"servo_use":1,"gpio":<gpio# 0-39>,"freq":<frequency Hz>}

現在の "pos" 値を取得する場合には "servo_pos" コマンドを使用してください。

error reply

{"result":"fail","error_text":"<error_message>"}

8.37 servo_pos

SERVO 位置を設定する。"ch"以外のパラメータを指定しない場合は現在の設定値を返す。

このコマンドを使用する前に必ず "servo_config" MGCPコマンドの "servo_use" パラメータで SERVO 使用を有効にしてください。

"servo_pos" コマンドで指定した SERVO位置 パラメータから PWM 周波数と分解能を元に PWM Duty値を計算します。詳しくは "servo_config" コマンドの項を参照してください。

"pos" は SERVO位置(サーボ信号パルス幅)を指定する。単位はマイクロ秒で 500(us) から 2500(us)の範囲で指定する。パラメータ省略時は現在の設定値を返しますが、リセット後に一度も "servo_pos" コマンドで SERVO位置を設定していない場合にはエラーになります。

通常のラジコン用SERVOの場合には、SERVO中間位置に対応するサーボパルス幅は 約1500us(1.5ms) になります。

SERVO製品ごとにサーボ信号パルス幅の変更で可動できる範囲が異なっています。事前にサーボの仕様を確認して、サーボパルス幅を仕様の範囲内に収まるように指定してください。

request

{"command":"servo_pos","ch":<0-3> [,"pos":<pos_us# 500-2500>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","ch":<ch# 0-3>,"pos":<pos_us# 500-2500>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.38 job_config

MGCPデバイスで繰り返しダイレクトコマンドを自動実行する JOB 機能の設定を行う。パラメータを１つも指定しない場合は現在の設定値を返す。

"mode" は JOB機能の動作モードを指定する。0,1,2 の値が指定可能でそれぞれの意味は下記になります。

0: JOB 機能を使用しない。このモードの場合には "job_once" コマンドも使用できません。(デフォルト値)

1: 定期的にダイレクトコマンドを繰り返し実行する。実行結果は全て破棄される。

2: 定期的にダイレクトコマンドを繰り返し実行する。実行結果ステータスとリターンパラメータは "job_exec" イベントで送信される。

"job_init","job_cmd" は JOB機能で繰り返し実行するダイレクトコマンドを指定する。デバイス起動後や "job_config" コマンドで設定値を変更した後の１回目は、"job_init" で指定したダイレクトコマンドを実行して、その後 "job_cmd" で指定したダイレクトコマンドを繰り返し実行する。"job_init","job_cmd" で指定するパラメータは、MGCP ダイレクトコマンドのコマンドとリクエストパラメータ部分から成るバイナリデータ列を16進数文字列形式で指定する。最大 50バイトまで指定可能。ダイレクトコマンドの詳細は "ダイレクトコマンド" の章を参照して下さい。

"interval" は コマンドの実行間隔をミリセコンド単位で指定する。-1 を指定した場合には自動的なコマンド繰り返しは行わない。-1 と 0 から 2147483647の値を指定できる。

"mode" パラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"job_config" [,"mode":<0|1|2>]

[,"job_init":"<hexstr>"][,"job_cmd":"<hexstr>"][,"interval":<ms# -1|0-2147483647>] }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","mode":<0|1|2>,

"job_init":"<hexstr>","job_cmd":"<hexstr>","interval":<ms# -1|0-2147483647>}

error reply

{"result":"fail","error_text":"<error_message>"}

8.39 job_once

JOB 機能で設定したダイレクトコマンドを実行間隔に関係なく 1 回実行させる。

デバイス起動後や "job_config" コマンドで設定値を変更した後に一回も JOB機能で設定したダイレクトコマンドを実行していなかった場合には、"job_config" コマンドの "job_init" パラメータで設定したダイレクトコマンドが実行されます。その後 "job_once" コマンド実行毎に "job_config" コマンドの "job_cmd" パラメータで設定したダイレクトコマンドを実行する。

"init" オプションに 0 を指定すると常に "job_cmd" パラメータで指定したダイレクトコマンドを実行して、1 を指定すると常に "job_init" パラメータで指定したダイレクトコマンドを実行する。

このコマンドを使用する前に、予め "job_config" コマンドの "mode" を 1 または 2 に設定して下さい。

"job_config" コマンドの "interval" に 0 以上の値が設定されている場合には、定期的な JOB コマンド実行に加えて、追加でダイレクトコマンドを実行します。-1 に設定していた場合には "job_once" 実行時のみダイレクトコマンドを実行します。

"job_once" で実行したダイレクトコマンド実行結果は、"mode" パラメータが 2 に設定されているときに限り "job_exec" イベントとして送信されます。

request

{"command":"job_once" [,"init":<0|1>]}

success reply (set)

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.40 dx_config

MGCPデバイス上で UDP サーバーを起動して、ダイレクトコマンドを実行する機能を設定する。パラメータを１つも指定しない場合は現在の設定値を返す。

コンフィギュレーションコマンド "dx_xxxx" で設定する内容と同じですので、コンフィギュレーションコマンド項の説明も参照して下さい。

"use" は UDPサーバー経由のダイレクトコマンド実行を行うかどうかを指定する。1:使用する, 0:使用しない。

"port" はダイレクトコマンド実行用に MGCPデバイスで作成されるUDPサーバーの UDP/IP ポート番号を指定する。

パラメータを指定して設定値を変更した場合には、続けて MGCP コマンド "save" を実行して変更した設定値をデバイスの不揮発領域に保存します。その後、MGCP "reset" コマンドを実行するかハードリセットを行うことで、新しい設定値で MGCP デバイスが動作します。

request

{"command":"dx_config" *[*,"use":<0|1>*]* *[*,"port":<udp_port# 1-65535>*]* }

success reply (set)

{"result":"success"}

success reply (get)

{"result":"success","use":<0|1>,"port":<udp_port# 1-65535>*]*}

error reply

{"result":"fail","error_text":"<error_message>"}

8.41 subscribe

指定したトピックの購読リクエストを MQTT ブローカに送信する。

MGCP デバイスはリクエストメッセージを取得(購読)するために、デフォルトでノード名と MAC アドレス、ブロードキャストアドレスに対応する topic を自動的に購読しています。このため、通常はこのコマンドを使用する必要はありません。

購読対象とするトピック名は "topic" タグに指定する。

ここで subscribe 設定をおこなったトピックは、デバイス内の不揮発領域には保存しないのでリセット後は自動的に購読対象にはなりません。

SUBSCRIBE メッセージ中の購読リクエスト QoS は 0 で固定です。

set request

{"command":"subscribe","topic":"<topic_string>"}

success reply

{"result":"success","error_text":""}

error reply

{"result":"fail","error_text":"<error_message>"}

8.42 nvs_set

コンフィギュレーション値を設定する。

nvs のキーと値のタイプ、値(数値または文字列)を指定して、MGCP デバイスのコンフィギュレーション領域に書き込む。キーと値のタイプが正しくない場合には、そのキーに対応する既存のコンフィギュレーション値は壊れます。

type タグに指定する文字列と nvs 保存時に使用するデータタイプの対応は以下になります。

type タグの値	nvs 保存時のタイプ	value タグの値
"u8"	uint8_t	整数 (0..255)
"i8"	int8_t	整数 (-128..127)
"u16"	uint16_t	整数 (0..65535)
"hex8"	uint8_t	16進数文字列2桁
"hex16"	uint16_t	16進数文字列4桁
"i32"	int32_t	整数
"hex32"	int32_t	16進数文字列8桁 (負値は設定できません、代わりに "i32" を使用してください)
"hex64"	uint64_t	16進数文字列16桁
"string"	char*	文字列
"blob"	uint8_t*	16進数文字列(バイト数 * 2桁)

set request

{"command":"nvs_set","type":<"string"|"u8"|"i8"|"u16"|"i32"|"hex8"|"hex16"|"hex32"|"hex64"|"blob">
,"key":"<tagname>","value":<number|"<string>"|"<hexstr>">}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.43 nvs_get

現在 MGCP デバイスに設定されているコンフィギュレーション値を取得する

nvs のキーと値のタイプ、MGCP デバイスのコンフィギュレーション値を取得する。キーと値のタイプが正しくない場合にはエラーになります。

セキュリティ上アクセス不可のキーを指定した場合にはエラーが返ります。(WiFi SSID,WiFi password,マスターパスワード等)

type タグに指定する文字列と nvs からリードするときに使用するデータタイプ、コマンド実行結果で得られる value タグの値は以下になります。

type タグの値	nvs 取得時のタイプ	value タグの値
"u8"	uint8_t	整数 (0..255)
"i8"	int8_t	整数 (-128..127)
"u16"	uint16_t	整数 (0..65535)
"hex8"	uint8_t	16進数文字列2桁
"hex16"	uint16_t	16進数文字列4桁
"i32"	int32_t	整数
"hex32"	int32_t	16進数文字列8桁 (負値のときはエラーになります、代わりに "i32" を使用してください)
"hex64"	uint64_t	16進数文字列16桁
"string"	char*	文字列
"blob"	uint8_t*	16進数文字列(バイト数 * 2桁)

query request

{"command":"nvs_get","type":<"string"|"u8"|"i8"|"u16"|"i32"|"hex8"|"hex16"|"hex32"|"hex64"|"blob">
,"key":"<tagname>"}

success reply

{"result":"success","value":<number|"<string>"|"<hexstr>">}

error reply

{"result":"fail","error_text":"<error_message>"}

8.44 nvs_init

nvs 保存領域を初期化する。

既存の全てのコンフィギュレーション値は破棄されます。このコマンド実行の後、必須のコンフィギュレーション値(WiFi SSID,MQTT Host等)を設定しない状態でデバイスをリセットすると、MGCP デバイスはコンソールモードで起動します。

set request

{"command":"nvs_init"}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

8.45 nvs_get_arr

複数のコンフィギュレーション値を取得する。

nvs のキー・プリフィックス文字列と個数を指定して、複数のバイト値から成る MGCP デバイスのコンフィギュレーション値を配列で取得する。キー・プリフィックスと個数が正しくない場合にはエラーになります。

MGCP デバイスの nvs に指定するキー文字列は <tagname_prefix>0,<tagname_prefix>1,<tagname_prefix>2, .. <tagname_prefix>(<count>-1) になります。

取得したコンフィギュレーション値は数値配列として得られます。配列要素の数値タイプは type パラメータで指定します。

<count> に指定可能な数は最大 50 までです。

例えば <tagname_prefix> が "i2c_" で、<count> に 2、タイプに "u8" を指定すると、"i2c_0", "i2c_1" の２つのキーを指定して nvs から 2 バイト分のデータを取得します。

query request

{"command":"nvs_get_arr","prefix":"<tagname_prefix>","count":<count>,"type":<"u8"|"i8"|"i32">}

success reply

{"result":"success","value":[<num#0,num#1, ... ,num#<count>-1]}

error reply

{"result":"fail","error_text":"<error_message>"}

8.46 nvs_set_arr

複数のコンフィギュレーション値を設定する。

nvs のキー・プリフィックス文字列と、数値配列を指定して、MGCP デバイスのコンフィギュレーション領域に書き込む。キー・プリフィックスと個数が正しくない場合には、そのキーに対応する既存のコンフィギュレーション値は壊れます。

value パラメータには数値配列として複数のコンフィギュレーション値を指定します。配列要素の数値タイプは type パラメータで指定します。また、指定可能な配列の最大要素数は 50 までです。

MGCP デバイスの nvs に指定するキー文字列は <tagname_prefix>0,<tagname_prefix>1,<tagname_prefix>2, .. <tagname_prefix>(<配列要素数>-1) になります。

例えば <tagname_prefix> が "i2c_" で、value に [val#0,val#1],タイプに "u8" を指定すると、"i2c_0", "i2c_1" の２つのキーを指定して nvs に val#0 と val#1 の2バイトのデータを設定します。

set request

{"command":"nvs_set_arr","prefix":"<tagname_prefix>"
,"type":<"u8"|"i8"|"i32">,"value":[<num#0,num#1, ... ,num#n]}

success reply

{"result":"success"}

error reply

{"result":"fail","error_text":"<error_message>"}

9　コンフィギュレーション・コマンド(コンソールコマンド)

MGCP デバイスは通常運用時に使用する MQTT ペイロードを利用したリモートモードの他に、MQTT ブローカへの接続情報や、GPIO, SPI, I2C バスのハードウエア設定等を行うためのコンソールモードを持っています。コンソールモードではコンフィギュレーションコマンドを使用して、デバイスの現在の設定値を確認したり内容を変更することができます。

Linux や Windows コンピュータ等に MGCP ファームウエアを書き込んだ ESP32デバイスを接続した後、(仮想)シリアルポート経由でターミナルエミューレータソフト等を使用して操作します。

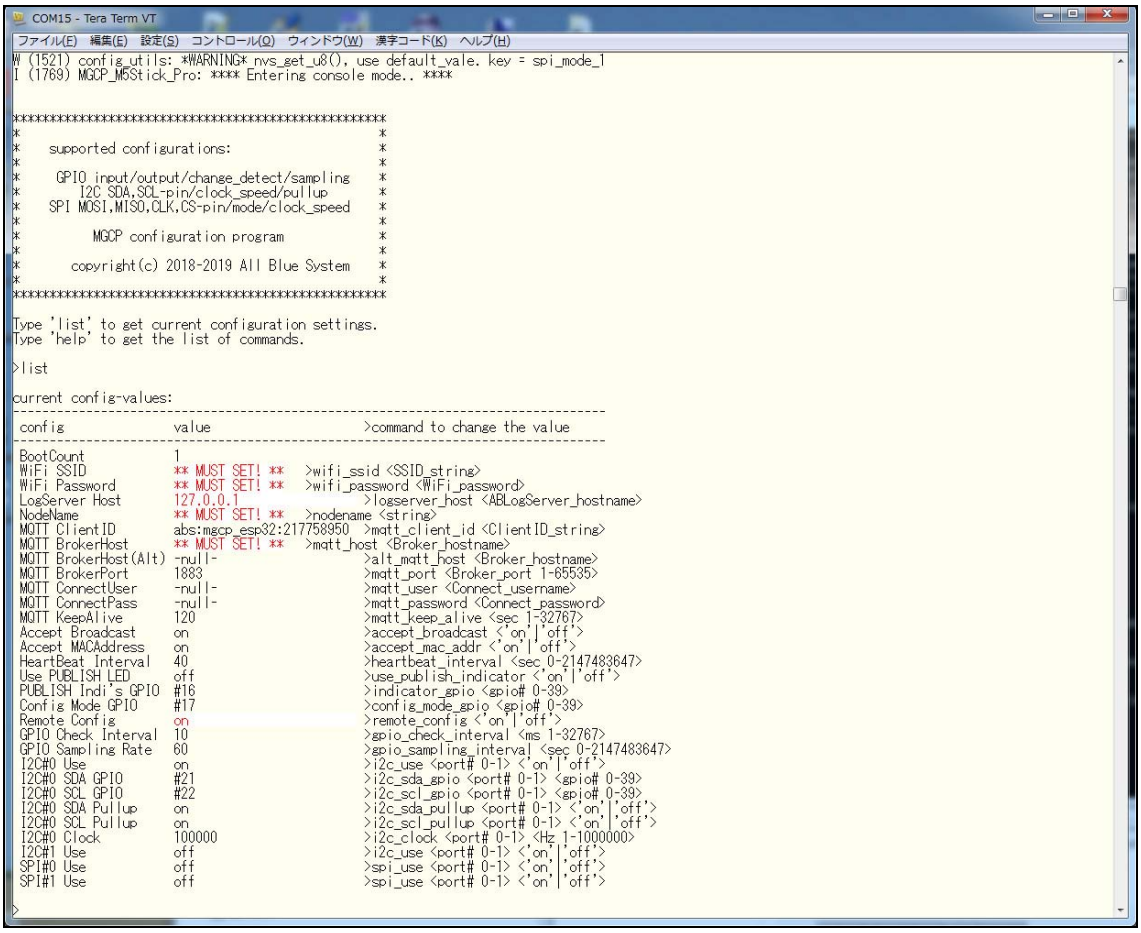
MGCP デバイスは下記の条件に一致する場合にコンソールモードで起動します。

(1) MGCP デバイス起動時に、"config_mode_gpio" コンフィギュレーション設定コマンドで指定した GPIOポートの値が Low になっていた場合。
"config_mode_gpio" のデフォルト値は "STD" デバイスタイプの場合は GPIO#17です。"M5Stack" デバイスタイプの場合は 正面にある３つのボタンの最も左側の "A"ボタン(GPIO#39)にアサインされています。 "M5Stick-C" デバイスタイプの場合は USBケーブルが左に見える位置の場合に下部にあるボタン(GPIO#39)にアサインされています。これらのポートやボタンを Power-on時に GND に接続(ボタンを押したまま)することで、コンフィギュレーションモードに入ります。

(2) MGCPデバイスノード名、MQTT ブローカホスト名、Wi-Fi SSID, WiFi-password の何れかのコンフィギュレーションが未設定の場合。
ESP32 デバイスに MGCP ファームウエアを書き込んだ後、最初に起動した時にはこれらの設定が行われていませんので必ずコンソールモードで起動します。

(3) MGCP コマンド "force_config" 実行によりデバイスをリセットさせて、MGCP デバイスが再起動した場合。

(4) デモライセンスで使用中に 30回以上リセット(再起動)した場合。
この場合には、コンソールモードで "reset_boot_counter" コマンドを実行することで引き続きデモライセンスで使用可能になります。



（ファームウエア書き込み直後に、コンソールモードで起動した時の様子）

コンソールモードでは ESP32 CPU ボードに付属のシリアルコンソールポートや USB 仮想COM ポート経由で接続します。Linux や Windows 上のターミナルソフトウエア等から接続しますが、詳しい接続方法については ESP32 プロセッサを搭載しているハードウエアのマニュアルを参照してください。例として "M5Stick-C" デバイスを接続した場合には下記の通信パラメータで接続します。

注意：ファームウエア書き込み時は、専用の Espressif Systems社製 "Flash download tools" を使用するため下記の通信条件とは異なっています。コンフィギュレーション・コンソールはファームウエア書き込み時と同じシリアルポートを使用しますが、ターミナルコンソール画面で手動でコマンド操作するとき専用の通信条件を使用します。

シリアルポート(コンソールモード)通信条件	
通信フォーマット	ASCII 文字, CR 終端(送受信共)
接続するデバイスファイル	/dev/ttyUSB0 や COM3 等
ボーレート	115200
ビット長	8
ストップビット	1
パリティ	無し
フロー制御	無し

コンソールモードで使用するコンフィギューレションコマンドを以下に説明します。

マニュアル中の command フォーマットの記述について

- コマンドとパラメータ間は１文字の(英字)スペース(0x20) で区切ってください。
- 日本語はコンフィギュレーション・コンソールコマンドでは使用できません。
- cmd <keyword> の記述の場合には <keyword> 部分に文字列を指定します。例：cmd param1
- cmd *[*<keyword>*]* の様にイタリック体の鍵括弧 *[ ]* でパラメータが囲まれている場合には、<keyword> 部分の文字列指定は省略可能です。

9.1 list

command

list

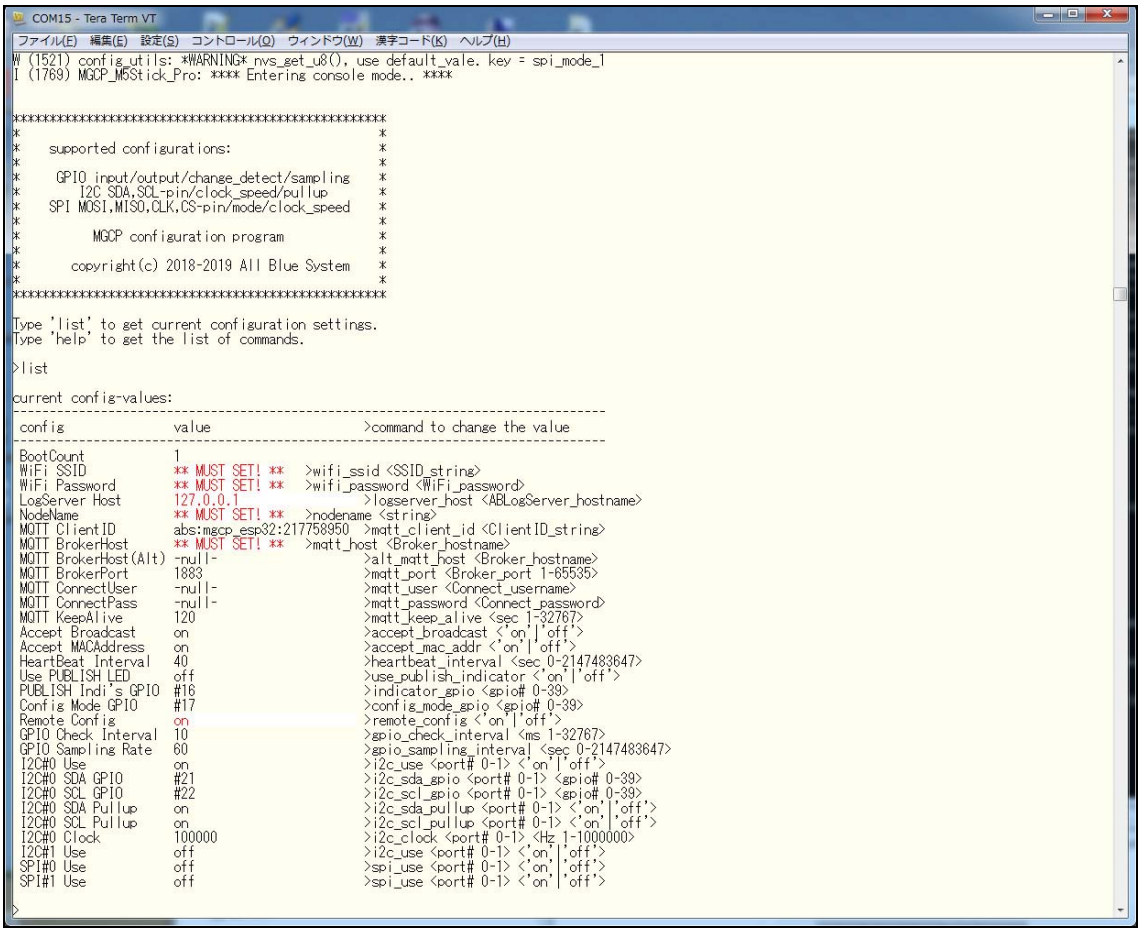
現在の設定値をコンソールに表示する。

コンフィギュレーションコマンドで設定値を変更した場合に "list" コマンドを実行すると、意図した値が設定されているかを簡単に確認できます。各コンフィギュレーション設定値の右に、設定を変更する場合のコマンド・ヘルプを表示します。

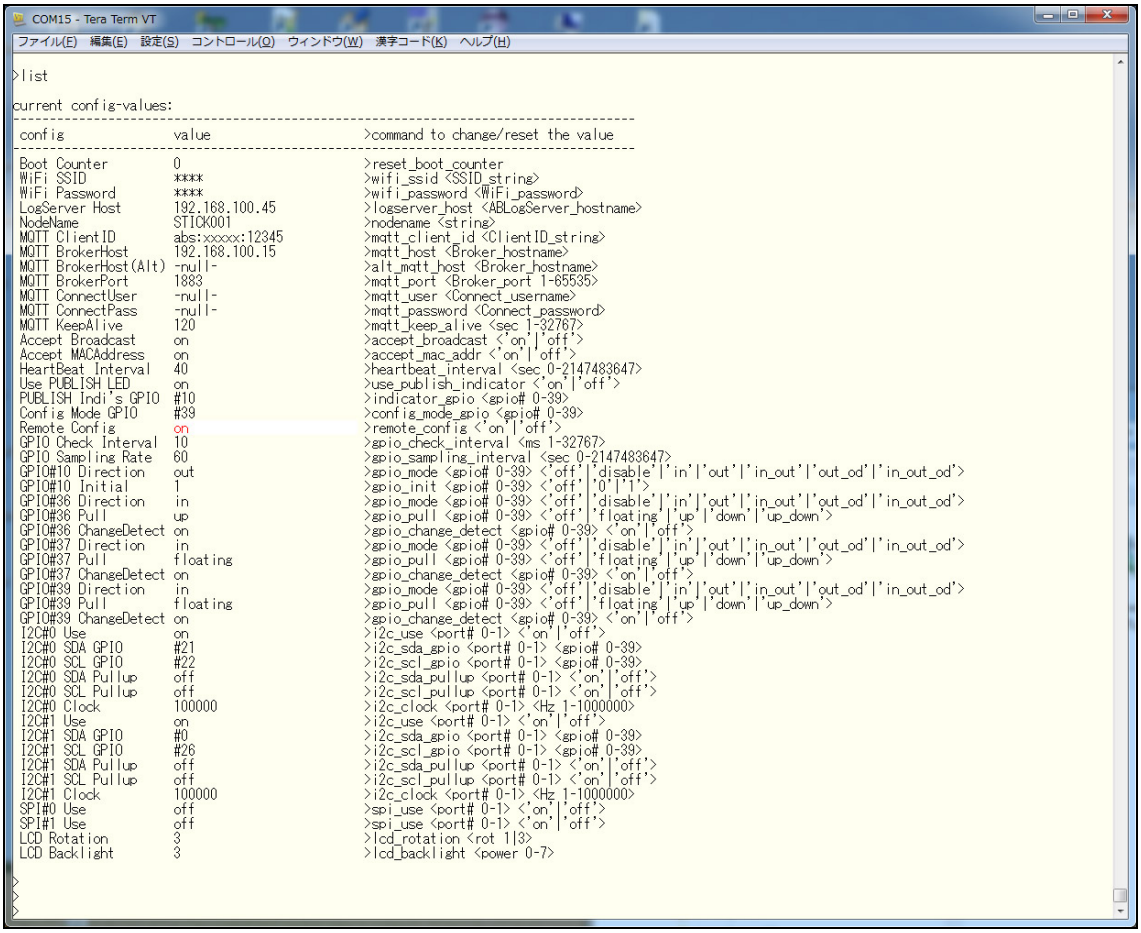
I2C や SPI, NeoPixel 関連の設定値は、該当するI2Cバス番号や SPI ポート,NeoPixel の "Use" フラグを "on" に設定することで詳しい設定内容が表示されます。

"list" コマンドで表示している設定値は "save" コマンドを実行するまではデバイス内部の不揮発領域には保存されていないので、保存前にリセットすると以前の設定値に戻りますので注意してください。

必須項目が設定されていない場合や設定値に注意が必要な場合には、設定項目を強調表示します。



（最初のファームウエア書き込み直後の実行例。未設定の必須設定項目は強調表示されます）



(Wi-Fi や MQTT ブローカ、I2C, GPIO 等を設置環境に合わせて設定した例)

9.2 help

command

help *[<keyword>]*

コンフィギューレションコマンド一覧を表示する。

<keyword> 文字列を指定すると、その文字列を先頭に含むコマンド一覧のみを表示する。

9.3 version

command

version

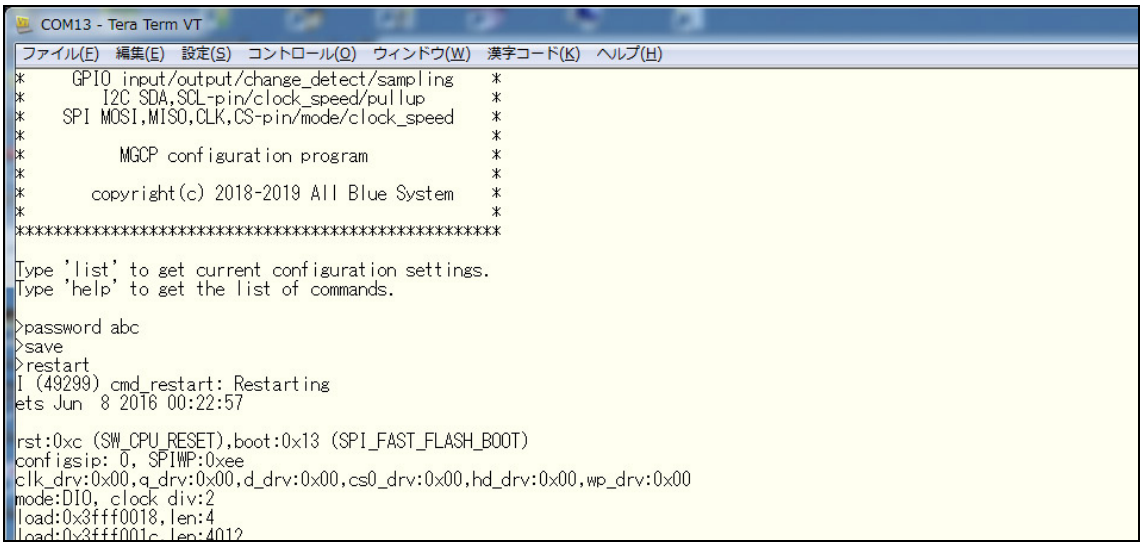
MGCP ファームウエアバージョン文字列を表示する。

9.4 password

command

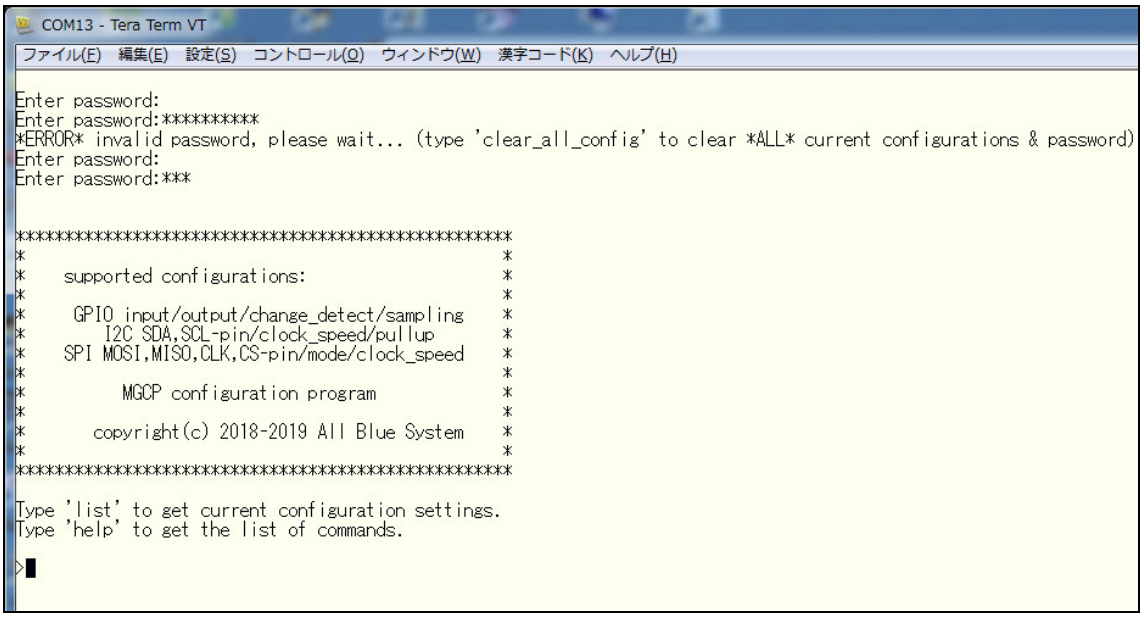
password <password_string>

コンソールモードに入るときのパスワード文字列を設定する。<password_string> にパスワード文字列を指定する。



（パスワード設定例）

このコマンド実行の後 "save" コマンドで不揮発領域に設定を保存すると、それ以降コンソールモードに入るときには、このコマンドで設定したパスワード入力を求められます。パスワードを間違えた場合には数秒のウェイトの後、再びパスワード入力モードに入ります。



（パスワード入力例）

設定したパスワード文字列はファームウエア更新時も、クリアしませんので注意してください。パスワードを忘れた場合に、パスワード文字列として "clear_all_config" を入力すると強制的にコンソールに入ることが可能です。ただしこの場合、MGCP デバイスに設定済みの既存のコンフィギィーレションは全て消去されデフォルト値に戻されます。

9.5 password_clear

command

password_clear

コンソールモードに入るときのパスワード文字列を消去する。

このコマンド実行の後 "save" コマンドで不揮発領域に設定を保存すると、その後コンソールモードに入るときにはパスワード入力は要求されなくなります。(デフォルト状態)

9.6 save

command

save

現在の設定値を不揮発領域に保存する。

コンフィギュレーション値を変更したときに "list" コマンドで変更内容を確認できますが、リセット後は最後に "save" コマンドを実行した時の状態に戻されます。

コンソールモードで設定値を変更した場合には "list" コマンドを実行して、意図した値に設定されていることを確認した後、"save" コマンドを実行して不揮発領域に保存して下さい。

例:

```
>wifi_ssid MYSSID_0123
>wifi_password 1234
>nodename STICK001
>list
(設定内容確認)
>save
>restart (またはコンソール端末を切断・終了後、デバイス電源 OFF->ON)
```

9.7 restart

command

restart

デバイスをリセットして MGCP ファームウエアを再起動させる。

MGCP デバイスの電源 OFF->ON やハードウエアリセットと同等の動作です。

9.8 clear_all_config

command

clear_all_config

MGCP デバイスに設定済みのコンフィギュレーションを全て消去する。

この後 "restart" コマンドを実行すると、デバイスのコンフィギュレーションはデフォルト値に設定される。

9.9 reset_boot_counter

command

reset_boot_counter

MGCP デバイスの起動された回数を示すカウンタを 0 にします。LCD を搭載している M5Stack, M5Stick-C 用の MGCP 起動時には 画面にカウンタ値が表示されます。また、MGCP デバイスが定期的に送信する "heartbeatbeat" イベントメッセージ中にもこのカウンタ値が格納されます。

デモ版の MGCP ファームウエアを使用している場合に、このコマンドを実行することでデモライセンスを延長できます。詳しくは "MGCPデバイス・インストール"の章を参照してください。

9.10 wifi_ssid

command

wifi_ssid *[<SSID_string>]*

MGCP デバイスが接続する Wi-Fi アクセスポイントの SSID 文字列を設定します。

MGCP デバイスの必須設定項目ですので、設置環境に合わせて適切な値を設定してください。

例：

```
>wifi_ssid MYSSID_0123
```

9.11 wifi_password

command

wifi_password *[<WiFi_password>]*

MGCP デバイスが接続する Wi-Fi アクセスポイントの認証用文字列を設定します。

MGCP デバイスの必須設定項目ですので、設置環境に合わせて適切な値を設定してください。

例：

```
>wifi_password MYPASSWORD
```

9.12 logserver_host

command

logserver_host *[<ABLogServer_hostname>]*

MGCP デバイスが出力するログメッセージの送信先アドレスを設定する。パラメータ省略時は現在の設定値を表示する。ABS-9000 LogServer を設置した Windows コンピュータの IP アドレスを設定します。複数の MGCP デバイスで同一の ABS-9000 LogServer を指定すると、ログ情報を一元化して管理することができます。

logserver_host で設定した ABS-9000 LogServer をインストールしたコンピュータが動作していない場合や、ネットワークが切断されている場合でも MGCPデバイスの動作には影響を与えません。

ログメッセージは MGCP デバイスのシリアルポートにも同時に出力していますので、仮想シリアルポートに接続することで同様のメッセージを取得することができます。

<ABLogServer_hostname> に "127.0.0.1" を設定すると、ログサーバーへのメッセージ出力を行いません。

例：

```
>logserver_host 192.168.100.45
```

9.13 nodename

command

nodename *[<string>]*

MGCP デバイスのノード名を設定する。パラメータ省略時は現在の設定値を表示する。

ノード名には英数文字のみ指定可能です。ここで設定するノード名はリクエスト先やイベント送出元の MGCP デバイスを区別するために使用します。同一の MQTT ブローカに接続する複数の MGCP デバイスには必ず各々ユニークなノード名を設定してください。

例：

```
>nodename STICK001
```

9.14 mqtt_client_id

command

mqtt_client_id *[<ClientID_string>]*

MGCP デバイスが MQTT ブローカに接続するときの ClientID 文字列を設定する。パラメータ省略時は現在の設定値を表示する。デフォルトではランダムな文字列を含む値が設定されますので、通常は変更する必要はありません。

<ClientID_string>には英数文字のみ指定可能です。

デバイスを再起動した場合にもここで設定した同じ ClientID を使用しますので、MQTT ブローカ側では同一デバイスからの接続リクエストとして受け付けられます。ただし、MGCP デバイス H/Wを入れ替えて ClientID を含むコン

フィギュレーション値を引き継いだ場合等には、MQTT ブローカ側での接続に問題が発生する場合があります。この場合には、別デバイスであることを知らせる為に新しいユニークな〈ClientID_string〉文字列に変更してください。

例：

```
>mqtt_client_id mgcp0001abc
```

9.15 mqtt_host

command

mqtt_host *[<Broker_hostname>]*

MGCP デバイスが接続する MQTT ブローカのホスト名または IP アドレスを設定する。パラメータ省略時は現在の設定値を表示する。ホスト名(ドメイン名)を指定した場合には、MGCP デバイスが Wi-Fi 接続時に IP アドレスを取得した DHCP サーバーで指定された DNS ホストを利用して名前の解決を試みます。
MGCP デバイスの必須設定項目ですので、設置環境に合わせて適切な値を設定してください。

例：

```
>mqtt_host 192.168.100.15
```

例：

```
>mqtt_host my_broker.somewhere.host.com
```

9.16 alt_mqtt_host

command

alt_mqtt_host *[<Broker_hostname>]*

MGCP デバイスが接続する予備の MQTT ブローカのホスト名または IP アドレスを設定する。パラメータ省略時は現在の設定値を表示する。

この設定を行った場合には、mqtt_host(または alt_mqtt_host) で設定したブローカとの通信が出来なくなった場合に、alt_mqtt_host(または mqtt_host) コマンドで指定したブローカを使用します。MGCP デバイス起動時には、最後に接続に成功した MQTT ブローカを最初に接続先として選択します。もし、MQTT ブローカとの通信ができなかった場合には、次に alt_mqtt_host (または mqtt_host)に指定したMQTT ブローカに接続を試みます。

例：

```
>alt_mqtt_host alt_broker.somewhere.host.com
```

9.17 alt_mqtt_host_clear

command

alt_mqtt_host_clear

alt_mqtt_host コマンドで設定した予備の MQTT ブローカホスト情報を削除する。

9.18 mqtt_port

command

mqtt_port *[<Broker_port 1-65535>]*

MQTT ブローカに接続するときの TCP/IP ポート番号を設定する。パラメータ省略時は現在の設定値を表示する。

9.19 mqtt_user

command

mqtt_user *[<Connect_username>]*

MQTT ブローカに接続するときのユーザー名を設定する。パラメータ省略時は現在の設定値を表示する。

9.20 mqtt_user_clear

command

mqtt_user_clear

mqtt_user コマンドで設定した接続ユーザー名を削除する。

9.21 mqtt_password

command

mqtt_password *[<Connect_password>]*

MQTT ブローカに接続するときのパスワード文字列を設定する。パラメータ省略時は現在の設定値を表示する。

9.22 mqtt_password_clear

command

mqtt_password_clear

mqtt_password コマンドで設定した接続パスワードを削除する。

9.23 mqtt_keep_alive

command

mqtt_keep_alive *[<sec 1-32767>]*

MQTT ブローカに接続するときの KeepAlive パラメータを設定する。パラメータ省略時は現在の設定値を表示する。

mqtt_keep_alive の約半分の間隔で、MGCP デバイスは PINGREQ メッセージを MQTT ブローカに定期的に送信します。

9.24 accept_broadcast

command

accept_broadcast *[<'on'|'off'>]*

MQTT ブローカ経由でリクエストコマンドを受け付ける時に、"_ALL_" で指定されたリクエスト先ノード名のメッセージをこの MGCP デバイスで受け付けるかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

9.25 accept_mac_addr

command

accept_mac_addr *[<'on'|'off'>]*

MQTT ブローカ経由でリクエストコマンドを受け付ける時に、MACアドレス(16進数文字列) で指定されたリクエスト先ノード名のメッセージをこの MGCP デバイスで受け付けるかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

9.26 heartbeat_interval

command

heartbeat_interval *[<sec 0-2147483647>]*

MGCP デバイスがイベントメッセージ "heartbeat" を送信する間隔を設定する。パラメータ省略時は現在の設定値を表示する。

9.27 use_publish_indicator

command

use_publish_indicator *[<'on'|'off'>]*

MGCP デバイスがリクエストメッセージを受信したときに、indicator_gpio コマンドで指定した GPIOポートを短時間ブリンク(on-off の繰り返し)表示させるかどうかを指定する。パラメータ省略時は現在の設定値を表示する。

例：

```
>use_publish_indicator on
```

9.28 indicator_gpio

command

indicator_gpio *[<gpio# 0-39>]*

コンフィギュレーションコマンド "use_publish_indicator" を "on" に設定したときに、ブリンク出力に使用する

GPIO ポート番号を設定する。パラメータ省略時は現在の設定値を表示する。

例：

```
>use_publish_indicator on
>gpio_mode 17 out
>gpio_pull 17 off
>gpio_init 17 0
>indicator_gpio 17
>save
```

9.29 config_mode_gpio

command

config_mode_gpio *[<gpio# 0-39>]*

MGCP デバイス起動時にコンフィギュレーション・コンソールの起動チェックを行う GPIO ポート番号を指定する。パラメータ省略時は現在の設定値を表示する。

このコマンドで指定した GPIO ポートは、MGCP デバイス起動時に "入力モード","内部プルアップ有効"に設定され、その後ポート値を取得します。ポート値が Low(GND) の場合にはコンフィギューレション・コンソールモードで MGCP デバイスが起動します。

GPIO#34-#39 を指定する場合に ESP32 内蔵プルアップ抵抗が存在しないので、オープン時の場合にはGPIO 値が常に Low になり、MGCP デバイスは常にコンソールモードで起動します。この場合にはデバイス外部にプルアップ抵抗を接続する必要がありますので注意してください。

9.30 remote_config

command

remote_config *[<'on'|'off'>]*

"nvs_xxxxx" で始まる MGCP コマンドをこのデバイスで有効にするかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

運用中にリモートからデバイスのコンフィギュレーションを設定・取得を行う場合には "on" に設定してください。

9.31 tx_power

command

tx_power *[<power# 0|40-82>]*

Wi-Fi の最大送信出力を設定する。パラメータ省略時は現在の設定値を表示する。

40(最低) から 82(最大) の範囲で設定する。0 を指定すると、デバイス初期値で決められたデフォルト値(最大出力)を使用する。

ここで設定した値を元に MGCP デバイス起動時に選択された "実際の" 最大送信出力は "startup" イベントデータに格納された "tx_power" タグの値を参照してください。

9.32 gpio_check_interval

command

gpio_check_interval *[<ms 1-32767>]*

GPIO 値の変化を検出するときに、チャタリングの影響を除去するためのチェック間隔を指定する。パラメータ省略時は現在の設定値を表示する。

9.33 gpio_sampling_interval

command

gpio_sampling_interval *[<sec 0-2147483647>]*

MGCP イベント "gpio_sampling" を送信する間隔を設定する。パラメータ省略時は現在の設定値を表示する。
0 を設定すると、MGCP イベント "gpio_sampling"は送信しない。

9.34 gpio_mode

command

gpio_mode <gpio# 0-39> *[<'off'|'disable'|'in'|'out'|'in_out'|'out_od'|'in_out_od'>]*

GPIOポートの入出力モードを設定する。パラメータ省略時は現在の設定値を表示する。
設定値の意味は下記になります。

"off":	MGCP プログラムで GPIO ポートのモード設定を変更しない。
"disable":	GPIO ポート無効
"in":	入力モード
"out":	出力モード
"in_out":	入出力モード
"out_od"	オープンドレイン出力モード
"in_out_od"	オープンドレイン入出力モード

例：

```
>gpio_mode 37 in
>gpio_pull 37 up
```

```
>gpio_change_detect 37 on
>save
```

⚠ **GPIO#34-#39 使用時の注意**

GPIO#34-#39 はプロセッサの仕様によって入力モードのみ使用できます。また、これらのGPIOポートは出力ドライバや内臓プルアップ・プルダウン回路は用意されていません。MGCP プログラムはこれらの GPIOポートに対してもESP-IDF API 経由でこれらのモード指定が可能ですがエラーが発生します。

参照：ESP32 Datasheet Version2.1

9.35 gpio_pull

command

gpio_pull <gpio# 0-39> *[<'off'|'floating'|'up'|'down'|'up_down'>]*

GPIOポートのプルアップを設定する。パラメータ省略時は現在の設定値を表示する。

設定値の意味は下記になります。

"off":	MGCP プログラムで GPIO プルアップを変更しない。
"floating":	フローティングモード
"up"	プルアップ
"down"	プルダウン
"up_down"	プルアップ＆プルダウン

例：

```
>gpio_mode 37 in
>gpio_pull 37 up
>gpio_change_detect 37 on
>save
```

⚠ **GPIO#34-#39 使用時の注意**

GPIO#34-#39 はプロセッサの仕様によって入力モードのみ使用できます。また、これらのGPIOポートは出力ドライバや内臓プルアップ・プルダウン回路は用意されていません。MGCP プログラムはこれらの GPIOポートに対してもESP-IDF API 経由でこれらのモード指定が可能ですがエラーが発生します。

参照：ESP32 Datasheet Version2.1

9.36 gpio_init

command

gpio_init <gpio# 0-39> *[<'off'|'0'|'1'>]*

MGCP デバイス起動時の GPIOポート初期値を設定する。パラメータ省略時は現在の設定値を表示する。

"indicator_gpio" コマンドで指定した GPIO ポートがブリンク表示を終了した時点で、GPIO ポートは gpio_init で設定した値になります。

設定値の意味は下記になります。

"off": MGCP プログラムでは GPIO 値の初期値を設定しない。

"0": Low 出力に設定する

"1" High 出力に設定する

例：

```
>use_publish_indicator on
>indicator_gpio 10
>gpio_mode 10 out
>gpio_init 10 1
>save
```

9.37 gpio_get

command

gpio_get <gpio# 0-39>

現在の GPIO ポート値を取得して、コンソールに表示する。

例：

```
>>gpio_get 10
 GPIO#10 Level        0                          >gpio_set <gpio# 0-39> <'0'|'1'>
>
```

9.38 gpio_set

command

gpio_set <gpio# 0-39> <'0'|'1'>

GPIO ポート値を指定した値に設定する。

例：

```
>gpio_set 10 0
```

9.39 gpio_change_detect

command

gpio_change_detect <gpio# 0-39> [<'on'|'off'>]

MGCP イベント "gpio_change_detect" で監視対象となる GPIO ポートを指定または解除する。パラメータ省略時は現在の設定値を表示する。

"on" を設定すると、MGCP イベント "gpio_change_detect"の監視対象になる。"off" で監視対象から外す。

例：

```
>gpio_mode 36 in
>gpio_pull 36 up
>gpio_change_detect 36 on
>save
```

9.40 i2c_use

command

i2c_use <port# 0-1> [<'on'|'off'>]

I2C ポートを使用するかどうかを設定する。パラメータ省略時は現在の設定値を表示する。
I2C ポートの詳細設定はこのコマンドを "on" に設定した後有効になる。

例：

```
>i2c_use 0 on
>i2c_sda_gpio 0 21
>i2c_scl_gpio 0 22
>i2c_sda_pullup 0 off
>i2c_scl_pullup 0 off
>i2c_clock 0 100000
>save
```

9.41 i2c_sda_gpio

command

i2c_sda_gpio <port# 0-1> [<gpio# 0-39>]

I2C ポートの SDAラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.42 i2c_scl_gpio

command

i2c_scl_gpio <port# 0-1> [<gpio# 0-39>]

I2C ポートの SCLラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.43 i2c_sda_pullup

command

i2c_sda_pullup <port# 0-1> *[<'on'|'off'>]*

I2C ポートの SDAラインのプルアップを設定する。パラメータ省略時は現在の設定値を表示する。

9.44 i2c_scl_pullup

command

i2c_scl_pullup <port# 0-1> *[<'on'|'off'>]*

I2C ポートの SCLラインのプルアップを設定する。パラメータ省略時は現在の設定値を表示する。

9.45 i2c_clock

command

i2c_clock <port# 0-1> *[<Hz 1-1000000>]*

I2C ポートの クロックスピード(周波数)を設定する。<Hz>パラメータ省略時は現在の設定値を表示する。

9.46 spi_use

command

spi_use <port# 0-1> *[<'on'|'off'>]*

SPI ポートを使用するかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

SPI ポートの詳細設定はこのコマンドを "on" に設定した後有効になる。

例：

```
>spi_use 0 on
>spi_miso_gpio 0 19
>spi_mosi_gpio 0 23
>spi_cs_gpio 0 14
>spi_clk_gpio 0 18
>spi_mode 0 0
>spi_clock 0 1000000
>save
```

9.47 spi_miso_gpio

command

spi_miso_gpio <port# 0-1> *[*<gpio# 0-39 or -1>*]*

SPI ポートの MISOラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.48 spi_mosi_gpio

command

spi_mosi_gpio <port# 0-1> *[*<gpio# 0-39 or -1>*]*

SPI ポートの MOSIラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.49 spi_clk_gpio

command

spi_clk_gpio <port# 0-1> *[*<gpio# 0-39 or -1>*]*

SPI ポートの CLKラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.50 spi_cs_gpio

command

spi_cs_gpio <port# 0-1> *[*<gpio# 0-39 or -1>*]*

SPI ポートの CSラインで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.51 spi_mode

command

spi_mode <port# 0-1> *[*<spi_mode 0-3>*]*

SPI ポートの 動作モードを設定する。<spi_mode>パラメータ省略時は現在の設定値を表示する。

9.52 spi_clock

command

spi_clock <port# 0-1> *[*<Hz 1-100000000>*]*

SPI ポートのクロックスピード(周波数)を設定する。<Hz>パラメータ省略時は現在の設定値を表示する。

9.53 neopixel_use

command

neopixel_use *[<'on'|'off'>]*

NeoPixel LED を使用するかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

各 NeoPixel LED の詳細設定はこのコマンドを "on" に設定した後有効になる。

例：

```
>neopixel_use on
>neopixel_data_gpio 17
>neopixel_count 37
>neopixel_brightness 20
>save
```

9.54 neopixel_data_gpio

command

neopixel_data_gpio *[<gpio# 0|2|4|5|12|13|14|15|16|17|18|19|21|22|23|25|26|27>]*

NeoPixel LED を接続する GPIO ポート番号を設定する。パラメータ省略時は現在の設定値を表示する。

9.55 neopixel_count

command

neopixel_count *[<1-1000>]*

"neopixel_data_gpio" コマンドで指定した GPIO ポートに接続している LED数を設定する。パラメータ省略時は現在の設定値を表示する。

9.56 neopixel_brightness

command

neopixel_brightness *[<0-255>]*

NeoPixel LED の明るさを設定する。パラメータ省略時は現在の設定値を表示する。

明るさの設定値は、MGCPプログラム中から使用している FastLED ライブラリの brightness と同等です。

9.57 neopixel_maxmilliamps

command

neopixel_maxmilliamps *[<mA 0-65535>]*

NeoPixel LED の最大消費電流の制限値を設定する。パラメータ省略時は現在の設定値を表示する。

0 を設定すると無制限になるが、この場合でも "neopixel_brightness" コマンドによって設定された明るさによって LED消費電流は制限されます。

9.58 pwm_use

command

pwm_use <ch# 0-3> *[<'on'|'off'>]*

PWM チャンネルを使用するかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

PWM/SERVO チャンネルの詳細設定はこのコマンドを "on" に設定した後有効になる。

例：

```
>pwm_use 0 on
>pwm_gpio 0 18
>pwm_resolution 0 13
>pwm_freq 0 5000
>pwm_duty 0 4096
>save
```

9.59 pwm_gpio

command

pwm_gpio <ch# 0-3> *[<gpio# 0-39>]*

PWM チャンネルで使用する GPIO ポート番号を設定する。<gpio>パラメータ省略時は現在の設定値を表示する。

9.60 pwm_resolution

command

pwm_resolution <ch# 0-3> *[<resolution# 0-20>]*

PWM 信号の Duty 値の分解能をビット幅で指定する。<resolution>パラメータ省略時は現在の設定値を表示する。

PWM Duty値は 0 から 2^resolution# - 1 間の範囲で指定可能になる。

⚠ **PWM 周波数と分解能にはトレードオフの関係があります**

PWM 周波数を高くする場合には、それに合わせて分解能を下げる必要があります。

例えばPWM周波数 5 KHz の場合の最大分解能は 13ビットになります。設定可能な最大周波数 40MHzの場合の分解能は 1ビットになります。詳しくは ESP32 プロセッサの "ESP32 Technical Reference Manual" 中の "LED PWM Controller" をご覧ください。

9.61 pwm_freq

command

pwm_freq <ch# 0-3> *[<frequency# Hz>]*

PWM 信号の周波数を指定する。<frequency>パラメータ省略時は現在の設定値を表示する。

9.62 pwm_duty

command

pwm_duty <ch# 0-3> *[<duty# 0-(2^resolution# - 1>]*

PWM 信号のDuty 値を指定する。<duty>パラメータ省略時は現在の設定値を表示する。

PWM Duty値は 0 から 2^resolution# - 1 間の範囲で指定する。

9.63 servo_use

command

servo_use <ch# 0-3> *[<'on'|'off'>]*

PWM チャンネルを SERVO 信号出力用に使用するかどうかを設定する。パラメータ省略時は現在の設定値を表示する。

PWM/SERVO チャンネルの詳細設定はこのコマンドを "on" に設定した後有効になる。

servo_use を "on" にすると、指定したチャンネルの PWM コンフィギュレーションが以下の値に設定される。

pwm_use <ch#> on
pwm_gpio <ch#> 現在の設定値 (初期値は -1 になっているので、続けて必ず "pwm_gpio" コマンドを使用して正しい GPIO番号を設定すること)
pwm_resolution <ch#> 16
pwm_freq <ch#> 50
pwm_duty <ch#> 4915　　(下記説明参照)

PWM で使用する GPIO ポート番号や SERVO 周波数を設定する場合には、"servo_use" コマンドに続いて該当する設定値をコンフィギュレーションコマンドで手動で変更してください。"servo_config" MGCP コマンドを使用すると、これらの設定値をリモートから一度に設定することができるので通常はそちらを使用してください。詳しくは "servo_config" MGCP コマンドの項を参照して下さい。

servo_use を "off" にすると、指定したチャンネルの PWM コンフィギュレーションが以下の値に設定される。

pwm_use <ch#> off

pwm_use を "on" に設定した時の pwm_duty のデフォルト値で設定される 4915 は、SERVO周波数50Hz が場合の SERVO 信号の中間位置を示します。この値は下記の様にして計算しています。

PWM分解能 16bit = 65536、SERVO 中間位置 = 1.5ms、SERVO 周波数 50(Hz) = 20(ms) とすると
(65536 - 1) * (1.5 / 20) = 4915 となる。

例：

```
>servo_use 3 on
>pwm_gpio 3 19
>save
```

9.64 dx_command

command

dx_command [<'on'|'off'>]

ダイレクトコマンド実行を行うための UDPサーバー機能を有効にするかどうかを設定する。パラメータ省略時は現在の設定値を表示する。UDPサーバーのポート番号を設定する "dx_port" コマンドは、このコマンドを "on" に設定した後有効になる。

例：

```
>dx_command on
>dx_port 9000
>save
```

9.65 dx_port

command

dx_port [<udp_port# 1-65535>]

ダイレクトコマンドを受け付ける UDPサーバーの UDP/IPポート番号を設定する。パラメータ省略時は現在の設定値を表示する。

10 MGCPコマンド・イベントのMQTT-Topic/Payloadフォーマット

この章は MGCPコマンド受信やリプライ送信、MGCPイベント送信時に使用しているデータフォーマットとプロトコルについて説明します。

10.1 MGCP(Megumino-Garden Communication Protocol)概要

このプロトコルは LANや PAN 等の近距離に配置された多数のデバイス(ノード)と、それらのデバイスを利用する複数のクライアントシステム(分散システム)を構築するための通信手順を規定しています。具体的なデバイス(ノード)としては自宅や庭・ガレージ、工場、オフィス等の Wi-Fiに接続されたセンサーやアクチュエータを想定しています。

MGCPデバイスやクライアントはイーサーネットや Wi-Fi 接続機能をもっていて、MQTT プロトコルで任意の MQTTブローカに接続できます。このとき複数の MGCPデバイスを同一の MQTT ブローカに接続可能で、それぞれのデバイスをノード名で区別して操作できます。

各 MGCP デバイスにはセンサーや I/O 装置が接続されていて、クライアントからのリクエストコマンドで操作することができます。リクエストコマンド・メッセージは MGCPデバイスが接続している MQTT ブローカ経由で配信します。

MGCP デバイスとクライアントの違いはリクエストコマンドを発行するかどうかだけの違いです。アクチュエータ等を接続した MGCP デバイスがクライアント機能を同時に持つこともできます。

リクエストメッセージに対応するリプライメッセージを MGCPデバイスからリクエスト元に送信することができます。このリプライメッセージも MQTT ブローカ経由でクライアント側にメッセージが届けられます。

複数の MGCP デバイスと複数のクライアントを同一の MQTT ブローカに接続することで、クライアントが複数の MGCP デバイスをコントロールしたり、MGCP デバイスが複数のクライアントからアクセスされて、これらを同時に行う様なシステムを実現できます。

MGCPデバイス側で発生する各種イベントを MQTTブローカーに配信する機能も有しています。イベントの種類には定期的に送信する heartbeat や GPIO が変化したことを通知する gpio_change_detect 等があります。

Wi-Fiネットワーク接続には TCP/IP 上の MQTT プロトコルを使用していて、この MQTT で定義されたペイロード送信とトピック選択機能を使って MGCP プロトコルを実現しています。

MGCP の特徴：

■ MQTT プロトコル(TCP/IP 接続)の上位レイヤに位置して、MQTT ペイロードとトピックを利用して MGCP プロトコルを実装

■ ネットワーク接続は MGCPデバイス側から MQTT ブローカ側への方向に接続されて、エラー発生の検出や再接続、デバイスの再起動も MGCP デバイス側で自動的に行う。このため、MGCPデバイス自身の IP アドレスや DNS サーバー設定値はダイナミックにアサインされたものを使用して、MGCP デバイスにリクエストコマンドを送信したりリプライやイベントを受信するクライアント側は、それらの情報について一切関知する必要がない。

■ クライアント側のデバイスやコンピュータは、操作対象の MGCP デバイスが接続されている MQTT ブローカに同様に接続する。クライアントは、リクエストメッセージを格納したペイロード(JSON文字列)とリクエスト先のデバイス・ノード名を含むトピック文字列を作成して MQTT ブローカーに送信する。このとき全デバイスを対象に送信することも可能。クライアントは同じ MQTT ブローカに複数同時に接続して各々のクライアントが MGCP デバイスを操作することができる。

■ ユーザーは MGCPデバイスに任意の名前(ノード名)を設定して、MGCPデバイスの区別を行うことができる。MGCP デバイスは自身に設定されたノード名のトピックを常に購読していて、自分宛のリクエストメッセージを選択して処理する。また、デフォルトでは MAC アドレス(16進数文字列)を利用したデバイス固有の名前のトピックやブロードキャストリクエスト用の "_ALL_" 文字列のトピックも常に購読していて、これら宛に配信されたリクエストメッセージも同様に処理できる。

■ デバイス側からリプライメッセージ(JSON文字列)をリクエスト元に返信する場合には、リクエストメッセージ受信時に指定されたトピック中のメッセージID 文字列を使用して、リプライメッセージ送信時のトピックに指定する。

■ デバイス側で特定の条件になったときには、イベントの種別を含んだトピックを使用してイベントメッセージ(JSON文字列)を送信する。

10.2 メッセージとトピック指定に使用する文字列

Payload は JSON 形式の文字列で記述する。

Payload中の JSON 文字列に日本語も使用可能であるが、この場合には UTF-8 で記述すること。
(但し、現在のバージョンの MGCP デバイスでは日本語文字列をパラメータに指定できるコマンドは無い)

Payload中の JSON フォーマットの内容は一部のタグをデフォルトで規定("command", "result", "error_text") している以外は、アプリケーション側で自由に決めることができる。

Payload に格納する **JSON 文字列は最大で 4096バイト**。MGCP デバイス側でのリソース制限から、これよりも少ないバイト数の Payload しか受け付けない場合がある。

ノード名は 50 バイト以内の任意の英数字を使用する。(日本語は使用できない)
MGCP デバイス側には設置時にシステム内でユニークなノード名の付与が必須です。クライアント側はノード名として自身のホスト名を流用しても構いません。クライアント側でもホスト名とは別に、システム内でユニークなノード名を使用できます。

MQTT PUBLISH パケット送信時に指定する **Topic 文字列は 256 バイト以内**に収まるように作成する。

10.3 リクエストメッセージ(リプライ無し)

Topic

＊ 全てのノードにリクエストメッセージを送信する

/request/_ALL_/<node_name_from>

＊ 特定のノードにリクエストメッセージを送信する

/request/<node_name_to>/<node_name_from>

Payload (JSON string)

{"command":"xxxxxx", ... }

"command" タグ:リクエスト内容(コマンド)を表す文字列を格納する。

コマンドの種類によって複数のパラメータをタグに含める場合がある。

その他のタグはアプリケーション毎に決めた任意の内容を格納する。

QoS

クライアント側 PUBLISH QoS は 0, 1, 2 の何れを指定しても可。

MGCP デバイス側からの SUBSCRIBE QoS は 0 で固定。

Topicを構成する文字列の説明

"/reuqest"	リクエストメッセージを示す文字列（固定）
"_ALL_"	全てのデバイス側で購読している事を期待したトピック名。(固定)
<node_name_to>	特定のノードに送信する場合の送信先ノード名。 デバイス側では常に自身のノード名のトピックを購読している事を期待する。
<node_name_from>	リクエストメッセージを送信したデバイスのノード名

10.4 リクエストメッセージ(リプライ要求)

Topic

* 特定のノードにリクエストメッセージを送信する

/request/<node_name_to>/<node_name_from>/<request_id>

(*) 全てのノードに同時にリプライが必要なリクエストメッセージを送信することは推奨しない。

(<node_name_to> に "_ALL_"を指定して、かつ <request_id> も指定する場合に相当)

ただし、<node_name_to> に指定したノード名で購読しているデバイスが複数存在していた場合(クライアント側のリクエストメッセージ発行時点では厳密には判らない)には、結果的に複数のデバイスにリプライメッセージ要求付きのリクエストをすることになる。この場合には、リクエスト元は最初にリプライメッセージを受信した内容をリクエスト結果として処理することになる。このように、複数のリプライメッセージを同一の <request_id> で受信した場合には最初に受信したものだけを処理して、その他のメッセージは無視するべきである。

Payload (JSON string)

{"command":"xxxxxx", ... }

"command" タグ:リクエスト内容(コマンド)を表す文字列を格納する。

コマンドの種類によって複数のパラメータをタグに含める場合がある。

その他のタグはアプリケーション毎に決めた任意の内容を格納する。

QoS

クライアント側 PUBLISH QoS は 0, 1, 2 の何れを指定しても可。

MGCP デバイス側からの SUBSCRIBE QoS は 0 で固定。

Topicを構成する文字列の説明

"/reuqest"	リクエストメッセージを示す文字列(固定)
<node_name_to>	特定のノードに送信する場合の送信先ノード名。 デバイス側では常に自身のノード名のトピックを購読している事を期待する。
<node_name_from>	リクエストメッセージを送信したデバイスのノード名
<request_id>	リクエストメッセージを特定するための任意の英数文字列。 文字列長は自由で、リクエストメッセージ毎にユニークな内容にすること。

10.5 リプライメッセージ

Topic

＊ リクエストメッセージに対応するリプライメッセージを送信する

/reply/<node_name_to>/<node_name_from>/<request_id>

Payload (JSON string)

{"result":<"success"|"fail">,"error_text":<error_message>,.... }

"result" タグ: リクエストの処理が正常終了した場合には"success" が設定され、エラー発生時には "fail" が設定される。

"error_text" タグ(オプション): リクエスト処理中のエラー内容が格納される。エラーがない場合には空文字列 "" を格納するか、または "error_text" タグごと省略可能。

その他のタグはアプリケーション毎に決めた任意の内容を格納できる

QoS

クライアント側 SUBSCRIBE QoS は 0, 1, 2 の何れを指定しても可。

MGCP デバイス側からの PUBLISH QoS は 0 で固定。

Topicを構成する文字列の説明

"/reply"	リプライメッセージを示す文字列(固定)
<node_name_to>	リプライメッセージ送信先ノード名。

リクエストメッセージを受信した時の <node_name_from> と同じ内容を設定する。
リクエスト発行元のデバイス側では、常に自身のノード名のトピックを購読している事を期待する。

<node_name_from>　リプライメッセージの送信元ノード名。自身のノード名を設定する。

<request_id>　リクエストメッセージで指定された <request_id> と同じ文字列を設定する。

10.6　イベントメッセージ

Topic

＊ 全てのノードにイベントを送信する

/event/_ALL_/<node_name_from>/<application_name>/<event_type_string>

＊ 特定のノードにイベントを送信

/event/<node_name_to>/<node_name_from>/<application_name>/<event_type_string>

Payload (JSON string)

{ .... }

アプリケーション毎に決めた任意の内容を格納する。

QoS

クライアント側 SUBSCRIBE QoS は 0, 1, 2 の何れを指定しても可。

MGCP デバイス側からの PUBLISH QoS は 1 で固定。

Topicを構成する文字列の説明

"/event"　イベントメッセージを示す文字列(固定)

"_ALL_"　全てのデバイス側で購読している事を期待したトピック名。(固定)

<node_name_to>　特定のノードに送信する場合の送信先ノード名。
受信するデバイス側では常に自身のノード名のトピックを購読している事を期待する。

<node_name_from>　イベントを送信したデバイスのノード名

<application_name>　送信元のデバイスで動作しているアプリケーションの名前(任意の文字列)を格納する。

<event_type_string>　イベントの種類を示す名前。イベント発生側で定義した任意のイベント名を格納する。

11　ダイレクトコマンド

この章は MGCPデバイスで使用可能なバイナリ形式のコマンド実行機能について説明します。

MGCP デバイスでは MQTT ブローカ経由の JSON フォーマットでやり取りするMGCPコマンドの他に、デバイスとクライアント間を Peer-To-Peer の UDP/IP 通信でバイナリコマンドを実行することができます。(以降、"ダイレクトコ

マンド"と略します)。ダイレクトコマンドは MGCPコマンドや MGCPイベントデータから、MGCPデバイスのノード情報(IPアドレス、UDPポート番号、ブートカウンタ)を把握している場合に、デバイスをより高速に操作したいときに利用できます。

MGCP デバイス上で作成するダイレクトコマンド用 UDP サーバーは MGCP デバイスのコンフィギュレーション・コマンド "dx_command"もしくは MGCP コマンド "dx_config" で有効にできます。デフォルトでは無効になっています。

ダイレクトコマンドの特徴は以下になります:

(1) バイナリフォーマットでコマンドとリプライが構成されているので、ネットワークでやりとりするデータサイズが少なく、構文解析処理が不要なため高速に動作する。

(2) MGCP デバイス上に作成されたUDP サーバーにクライアントから直接コマンドを送信して、リプライも直接サーバーからクライアントに送信される。ブローカ・サーバーを経由しないため高速に動作する。(Peer-To-Peer通信)

(3) MGCP デバイス自身の IP アドレスはダイナミックにアサインされるため、クライアント側では MGCP イベントデータ "heartbeat"を購読して、最新の IP アドレスとダイレクトコマンド用の UDPサーバー・ポート番号、ダイレクトコマンド認証に必要なブートカウンタを取得する必要がある。このため、MQTT ブローカで認証できないクライアントはダイレクトコマンド実行に必要な情報を得られない。

(4) 高速動作のためダイレクトコマンドの認証は 1バイト長のブートカウントのみの簡単な仕組みで実現している。このため、セキュリティが確保された LAN 内でのみダイレクトコマンドを使用すること。またダイレクトコマンドに使用する UDP/IPポートは外部(WAN,Internet等)との通信経路をルータ等で適切に遮断してください。

⚠ **LAN内の MGCPデバイスは外部からダイレクトコマンドで操作できません。**

MGCPデバイスにアサインされた IPアドレスとUDPポートを指定することでダイレクトコマンドによるデバイス操作が可能ですが、操作対象の MGCP デバイスがクライアント側からネットワークアクセス接続可能な場合に限ります。例えば、MQTTブローカへのアクセスが MGCPデバイスとクライアント双方から可能でも、MGCPデバイスがルータ装置で区切られた LAN内のプライベートアドレスで動作している場合には操作できない場合があります。但しこの場合でも MGCP コマンドによる操作は常に可能です。

11.1 リクエストデータとリプライデータ

ダイレクトコマンドは１バイトのコマンド種別と、オプションで複数バイトから成るパラメータで構成されています。(下図参照)

(ダイレクトコマンド・リクエストデータ)

```
<Command> コマンド種別 1Byte (固定)
<Param#0> オプションパラメータ
<Param#1>
```

. .
<Param#n>

上記のダイレクトコマンドを実行した結果はリプライデータとして返ってきて、１バイトのステータスコードとオプションで複数バイトから成るリターンデータで構成されます。(下図参照)

(ダイレクトコマンド・リプライデータ)

<Status> ステータスコード 1Byte (固定) 0: Error, 1:Success
<Data#0> リターンデータ
<Data#1>
. .
<Data#n>

上記のオプションパラメータとリターンデータはダイレクトコマンドによって内容が異なります。オプションパラメータやリターンデータが無いダイレクトコマンドもあります。

ダイレクトコマンドは JOB実行機能でも利用できます

ダイレクトコマンドは MGCPデバイスの JOB機能でも使用できます。JOB機能については "job_config" MGCPコマンドを参照してください。

11.2 UDPデータパケットフォーマット

ダイレクトコマンドを MGCP デバイスの UDP サーバーに送信する時には、ダイレクトコマンドのリクエストデータ(コマンド種別 + オプションパラメータ)にヘッダとチェックサムを付加したリクエストパケットを作成して送信します。MGCPデバイス側でダイレクトコマンドを処理した結果は、ダイレクトコマンドのリプライデータ(ステータスコード + リターンデータ)にヘッダとチェックサムを付加したリプライパケットがリクエスト元に送信されます。

各パケットデータの構造は下記になります。リクエストパケットデータとリプライパケットデータの**最大サイズは1024バイト**です。この制限はダイレクトコマンド自身のサイズ制限ではなく、UDPパケットに格納するダイレクトコマンドのみに適用されます。

(リクエストコマンドパケット・データ構造)

offset	item	(request packet)
0	<StartDelimiter>	0x7E
1	<Length MSB>	<BootCounter> から <Params#n> までのバイト数
2	<Length LSB>	
3	<BootCounter>	下位 1byte分のみチェック
4	<MessageID>	リプライ無しの場合は 0x00, リプライ要求の場合は 0x00 以外の値を格納

5	<Command>	コマンド種別
(6)	<Param#0>	コマンドパラメータ#0(必要な場合のみ)
(7)	<Param#1>	コマンドパラメータ#1(必要な場合のみ)
..		
..		
(xx)	<Param#n>	コマンドパラメータ#n(必要な場合のみ)
xx	<CheckSum>	<BootCounter> から <Params#n> までのチェックサム値

(リプライパケット・データ構造)

offset	item	(reply packet)
0	<StartDelimiter>	0x7E
1	<Length MSB>	<MessageID> から <Data#n> までのバイト数
2	<Length LSB>	
3	<MessageID>	Request Packet 中に指定された<MessageID> をそのまま格納, 0x00 以外の値
4	<Status>	実行結果(0: error, 1:success)
(5)	<Data#0>	リプライデータ#0(必要な場合のみ)
(6)	<Data#1>	リプライデータ#1(必要な場合のみ)
..		
..		
(xx)	<Data#n>	リプライデータ#n(必要な場合のみ)
xx	<CheckSum>	<MessageID> から <Data#n> までのチェックサム値

以降の項では、各ダイレクトコマンドのコマンド種別とリクエストパラメータ、ステータスとリターンデータについて記述しています。UDPパケットデータ作成時に必要なヘッダ情報とチェックサム部分は除いて記述していますので注意して下さい。

ダイレクトコマンドのパラメータについては、同様の機能を実行するMGCP コマンドがある場合には各 MGCPコマンドのパラメータ説明を参照してください。

マニュアル中のダイレクトコマンドフォーマットの記述について

- 括弧 <xxxx> で記載されている各データの１行は 1 バイト分のデータを示します。
- MSB は整数値の最上位バイトを示し、LSB は最下位バイトを示します。16ビット以上の整数の場合にはMSB と LSB の間に8 ビット間隔でそのビット位置の値を示すパラメータやデータ値が挿入されます。
- <Command> はダイレクトコマンド種別を表す 1 バイトデータで必須です。
- <Param#x> はダイレクトコマンドのリクエストパラメータでコマンド毎に決められています。

● *[*<Param#x>*]* の様にイタリック体の鍵括弧 *[ ]* でリクエストパラメータが囲まれている場合には省略可能なパラメータであることを示します。

● <Status> はダイレクトコマンドの実行結果ステータスを表す 1 バイトデータで必須です。

● <Data#x> はダイレクトコマンドのリターンデータでコマンド毎に決められています。

● *[*<Data#x>*]* の様にイタリック体の鍵括弧 *[ ]* でリターンデータが囲まれている場合には実行結果によって省略されることがあるデータであることを示します。

11.3 verbose

ダイレクトコマンド関連のデバッグログ出力レベルの設定

--------- set verbose mode ---------		
<Command>	0x01	
<Param#0>	level 0-2	
--------- reply ----------		
<Status>	0(error) or 1(success)	

11.4 echo

パラメータで渡されたデータをそのまま返す

--------- echo ---------		
<Command>	0x02	
[<Param#0>*]*	data#0	任意のバイトデータ#0
[<Param#1>*]*	data#1	任意のバイトデータ#1
..		
[<Param#n>*]*	data#n	任意のバイトデータ#n
--------- reply ---------		
<Status>	0(error) or 1(success)	
[<Data#0>*]*	data#0	
[<Data#1>*]*	data#1	
..		
[<Data#n>*]*	data#n	

11.5 nodename

ノード名取得

--------- nodename ---------		
<Command>	0x0E	
--------- reply ---------		
<Status>	0(error) or 1(success)	
<Data#0>	char#0 (null terminated string)	
<Data#1>	char#1	

```
<Data#2>            char#2
..
<Data#n>            0x00
```

11.6 version

ファームウエアバージョン情報取得

```
--------- version ---------
<Command>           0x0F
--------- reply ---------
<Status>            0(error) or 1(success)
<Data#0>            char#0 (null terminated string)
<Data#1>            char#1
<Data#2>            char#2
..
<Data#n>            0x00
```

11.7 sequence

複数のダイレクトコマンドを実行

ダイレクトコマンドは各コマンド毎に１つのコマンド種別(1バイトデータ)が割り当てられていますが、"sequence" ダイレクトコマンドを使用すると複数のダイレクトコマンドを１つのダイレクトコマンドに纏めることができます。この場合に、複数のダイレクトコマンド&リクエストパラメータは全て "sequence" コマンドのリクエストパラメータとして指定します。また、パラメータで指定する複数のダイレクトコマンドの実行間隔(ディレイ)を設定することができます。

複数のダイレクトコマンドの実行結果はそれぞれの実行結果ステータスとリターンデータが纏まられて、"sequence" ダイレクトコマンドのリターンデータに格納されます。

"sequence" ダイレクトコマンドのパラメータで指定する複数のダイレクトコマンドに、"sequence" ダイレクトコマンドを"入れ子"の様に格納することもできます。

```
------------ sequence command -------------
<Param> にはコマンドとパラメータを格納したものを任意の個数分設定できる。
各コマンドの前にはそのコマンドとパラメータ部分の長さ(フレームサイズ)が入る。
Delay パラメータには、各コマンド実行間のディレイ時間を設定することができる。(0 でディレイ無し)
シーケンス内のコマンドにシーケンスコマンドを含めることもできる。

<Command>  0x0D
<Param#0>    Inter-Command-Delay MSB        Item# 間のコマンド実行間隔 [ms]
```

```
<Param#1>                        LSB
<Param#2>    Item#0 Length MSB  Item#0 <Command> から Item#0 <Param#n> までのフレームサイズ
<Param#3>                LSB
<Param#4>    Item#0 <Command>
[<Param#5>]  Item#0 <Param#0>
[<Param#6>]  Item#0 <Param#1>
          ..
[<Param#x>]  Item#0 <Param#n>
          ..
[<Param#x>]  Item#m Length MSB  Item#m <Command> から Item#m <Param#n> までのフレームサイズ
[<Param#x>]                LSB
[<Param#x>]  Item#m <Command>
[<Param#x>]  Item#m <Param#0>
[<Param#x>]  Item#m <Param#1>
          ..
[<Param#x>]  Item#m <Param#n>
------------  reply --------------------
<Status>     0(error)シーケンス中のコマンド実行時にエラーを検出した。
                     シーケンス内の一部のコマンドが成功した場合も含む。
             1(success) シーケンスに含まれる全てのコマンド実行が成功した。
                     この時、Item#0 <Status> .. Item#m <Status> の全ての値は 1 になっている。
<Data#0>     Item#0 Length MSB Item#0 <Status> から Item#0 <Data#n> までのフレームサイズ
<Data#1>                 LSB
<Data#2>     Item#0 <Status>
[<Data#3>]   Item#0 <Data#0>
[<Data#4>]   Item#0 <Data#1>
          ..
[<Data#x>]   Item#0 <Data#n>
          ..
[<Data#x>]   Item#m Length MSB Item#m <Status> から Item#m <Data#n> までのフレームサイズ
[<Data#x>]                 LSB
[<Data#x>]   Item#m <Status>
[<Data#x>]   Item#m <Data#0>
[<Data#x>]   Item#m <Data#1>
          ..
[<Data#x>]   Item#m <Data#n>
```

11.8 (LCD)fill screen

LCD画面を指定した色で塗りつぶす

```
--------- clear screen ---------
<Command>            0x10
<Param#0>            color code
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.9 (LCD)draw line

LCD画面に線を描画

```
--------- draw line ---------
<Command>            0x11
<Param#0>            [line#0] x0 MSB
<Param#1>            [line#0]    LSB
<Param#2>            [line#0] y0 MSB
<Param#3>            [line#0]    LSB
<Param#4>            [line#0] x1 MSB
<Param#5>            [line#0]    LSB
<Param#6>            [line#0] y1 MSB
<Param#7>            [line#0]    LSB
<Param#8>            [line#0] color code
..
..
[<Param#x>]          [line#n] x0 MSB
[<Param#x>]          [line#n]    LSB
[<Param#x>]          [line#n] y0 MSB
[<Param#x>]          [line#n]    LSB
[<Param#x>]          [line#n] x1 MSB
[<Param#x>]          [line#n]    LSB
[<Param#x>]          [line#n] y1 MSB
[<Param#x>]          [line#n]    LSB
[<Param#x>]          [line#n] color code
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.10 (LCD)draw circle

LCD画面に円を描画

```
--------- draw circle ---------
<Command>            0x12
<Param#0>            [circle#0] fill 0:off, 1:on
<Param#1>            [circle#0] x  MSB
```

```
<Param#2>            [circle#0]    LSB
<Param#3>            [circle#0] y  MSB
<Param#4>            [circle#0]    LSB
<Param#5>            [circle#0] r  MSB
<Param#6>            [circle#0]    LSB
<Param#7>            [circle#0] color code
..
..
[<Param#x>]          [circle#n] fill 0:off, 1:on
[<Param#x>]          [circle#n] x  MSB
[<Param#x>]          [circle#n]    LSB
[<Param#x>]          [circle#n] y  MSB
[<Param#x>]          [circle#n]    LSB
[<Param#x>]          [circle#n] r  MSB
[<Param#x>]          [circle#n]    LSB
[<Param#x>]          [circle#n] color code
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.11 (LCD)draw triangle

LCD画面に三角を描画

```
--------- draw triangle ---------
<Command>            0x13
<Param#0>            [triangle#0] fill 0:off, 1:on
<Param#1>            [triangle#0] xa MSB
<Param#2>            [triangle#0]    LSB
<Param#3>            [triangle#0] ya MSB
<Param#4>            [triangle#0]    LSB
<Param#5>            [triangle#0] xb MSB
<Param#6>            [triangle#0]    LSB
<Param#7>            [triangle#0] yb MSB
<Param#8>            [triangle#0]    LSB
<Param#9>            [triangle#0] xc MSB
<Param#10>           [triangle#0]    LSB
<Param#11>           [triangle#0] yc MSB
<Param#12>           [triangle#0]    LSB
<Param#13>           [triangle#0] color code
..
..
```

/<Param#x>/	[triangle#n] fill 0:off, 1:on
/<Param#x>/	[triangle#n] xa MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] ya MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] xb MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] yb MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] xc MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] yc MSB
/<Param#x>/	[triangle#n] LSB
/<Param#x>/	[triangle#n] color code
--------- reply ---------	
<Status>	0(error) or 1(success)

11.12 (LCD)draw rect

LCD画面に四角を描画

--------- draw rect ---------	
<Command>	0x14
<Param#0>	[rect#0] fill 0:off, 1:on
<Param#1>	[rect#0] x MSB
<Param#2>	[rect#0] LSB
<Param#3>	[rect#0] y MSB
<Param#4>	[rect#0] LSB
<Param#5>	[rect#0] w MSB
<Param#6>	[rect#0] LSB
<Param#7>	[rect#0] h MSB
<Param#8>	[rect#0] LSB
<Param#9>	[rect#0] color code
..	
..	
/<Param#x>/	[rect#n] fill 0:off, 1:on
/<Param#x>/	[rect#n] x MSB
/<Param#x>/	[rect#n] LSB
/<Param#x>/	[rect#n] y MSB
/<Param#x>/	[rect#n] LSB
/<Param#x>/	[rect#n] w MSB

```
/[<Param#x>]          [rect#n]    LSB
/[<Param#x>]          [rect#n] h  MSB
/[<Param#x>]          [rect#n]    LSB
/[<Param#x>]          [rect#n] color code
--------- reply ---------
<Status>              0(error) or 1(success)
```

11.13 (LCD)print ascii string

LCD画面に英数字を描画

```
--------- print ascii string ---------
<Command>             0x15
<Param#0>             wrap       0:off, 1:on
<Param#1>             cls_color  0-29,  255:no clear screen
<Param#2>             font       0-9,   255:use default
<Param#3>             fg_color   0-29,  255:use default
<Param#4>             bg_color   0-29,  255:use default
<Param#5>             x_pos  MSB         255:center position
<Param#6>                    LSB
<Param#7>             y_pos  MSB         255:center position
<Param#8>                    LSB
<Param#9>             char#0 (null terminated string)
<Param#10>            char#1
<Param#11>            char#2
..
..
<Param#xx>            0x00
--------- reply ---------
<Status>              0(error) or 1(success)
```

11.14 (LCD)draw bitmap

LCD画面にビットマップ図形を描画

```
--------- draw bitmap ---------
<Command>             0x16
<Param#0>             cls_color  0-29,  255:no clear screen
<Param#1>             fg_color   0-29,  255:use default
<Param#2>             bg_color   0-29,  255:use default
<Param#3>             mag_size   1-10,  255:use default
<Param#4>             [bitmap#0] x       MSB
<Param#5>                                LSB
```

```
<Param#6>           [bitmap#0] y      MSB
<Param#7>                             LSB
<Param#8>           [bitmap#0] w      MSB
<Param#9>                             LSB
<Param#10>          [bitmap#0] h      MSB
<Param#11>                            LSB
<Param#12>          [bitmap#0] byte#1
<Param#13>          [bitmap#0] byte#2
<Param#14>          [bitmap#0] byte#3
            ..
<Param#xx>          [bitmap#0] byte#x: x = h * ((w + 7)/8)
..
..
[<Param#xx>]        [bitmap#n] x      MSB
[<Param#xx>]                          LSB
[<Param#xx>]        [bitmap#n] y      MSB
[<Param#xx>]                          LSB
[<Param#xx>]        [bitmap#n] w      MSB
[<Param#xx>]                          LSB
[<Param#xx>]        [bitmap#n] h      MSB
[<Param#xx>]                          LSB
[<Param#xx>]        [bitmap#n] byte#1
[<Param#xx>]        [bitmap#n] byte#2
[<Param#xx>]        [bitmap#n] byte#3
            ..
[<Param#xx>         [bitmap#n] byte#x: x = h * ((w + 7)/8)
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.15 (LCD)draw pixel

LCD画面に点を描画

```
--------- draw pixel ---------
<Command>           0x17
<Param#0>           [pixel#0] x MSB
<Param#1>           [pixel#0]   LSB
<Param#2>           [pixel#0] y MSB
<Param#3>           [pixel#0]   LSB
<Param#8>           [pixel#0] color code
..
```

```
..
[<Param#x>]          [pixel#n] x MSB
[<Param#x>]          [pixel#n]   LSB
[<Param#x>]          [pixel#n] y MSB
[<Param#x>]          [pixel#n]   LSB
[<Param#x>]          [pixel#n] color code
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.16 (GPIO)gpio set

GPIOポート値の設定

```
--------- gpio set ---------
<Command>            0x41
<Param#0>            [port#0] gpio port   0-39
<Param#1>            [port#0] gpio level 0 or 1
[<Param#2>]          [port#1] gpio port   0-39
[<Param#3>]          [port#1] gpio level 0 or 1
..
..
[<Param#x>]          [port#x] gpio port   0-39
[<Param#x>]          [port#x] gpio level 0 or 1
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.17 (GPIO)gpio get

GPIOポート値の取得

```
--------- gpio get ---------
<Command>            0x40
<Param#0>            [port#0] gpio port   0-39
[<Param#1>]          [port#1] gpio port   0-39
..
[<Param#X>]          [port#n] gpio port   0-39
--------- reply ---------
<Status>             0(error) or 1(success)
<Data#0>             gpio level#0
[<Data#1>]           gpio level#1
..
[<Data#X>]           gpio level#n
```

11.18 (I2C)i2c read

I2Cスレーブデバイスからデータ読み込み

```
--------- i2c read ---------
<Command>           0x27
<Param#0>           port                i2c port number
<Param#1>           slave_addr          7bit i2c-slave address
<Param#2>           read_len MSB        read_len(bytes) > 0
<Param#3>                   LSB
--------- reply ---------
<Status>            0(error) or 1(success)
<Data#0>            read_data#0
<Data#1>            read_data#1
..
<Data#X>            read_data#n
```

11.19 (I2C)i2c write

I2Cスレーブデバイスにデータ書き込み

```
--------- i2c write ---------
<Command>           0x25
<Param#0>           port                i2c port number
<Param#1>           slave_addr          7bit i2c-slave address
<Param#2>           write_data#0
[<Param#3>]         write_data#1
..
[<Param#X>]         write_data#n
--------- reply ---------      (read_len == 0)
<Status>            0(error) or 1(success)
```

11.20 (I2C)i2c write and read

I2Cスレーブデバイスにデータ書き込みと読み込み

```
--------- i2c write and read ---------
<Command>           0x26
<Param#0>           port                i2c port number
<Param#1>           slave_addr          7bit i2c-slave address
<Param#2>           read_len MSB        read_len(bytes) > 0
<Param#3>                   LSB
<Param#4>           delay   MSB         delay == 0 : no read delay
<Param#5>                   LSB         delay > 0  : read delay(ms)
```

```
<Param#6>            write_data#0
[<Param#7>]          write_data#1
..
[<Param#X>]          write_data#n
--------- reply ---------
<Status>             0(error) or 1(success)
<Data#0>             read_data#0
<Data#1>             read_data#1
..
<Data#X>             read_data#n
```

11.21 (JOB)job once

JOB 機能で設定したダイレクトコマンドを実行間隔に関係なく１回実行させる

```
--------- job once ---------
<Command>            0x60
<Param#0>            init                0 or 1 (option)
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.22 (Neopixel)draw pixel rgb

指定したLEDの色をRGBで設定

```
--------- draw pixel rgb ---------
<Command>            0x34
<Param#0>            [pixel#0] x MSB
<Param#1>            [pixel#0]   LSB
<Param#2>            [pixel#0] r
<Param#3>            [pixel#0] g
<Param#4>            [pixel#0] b
..
..
[<Param#x>]          [pixel#n] x MSB
[<Param#x>]          [pixel#n]   LSB
[<Param#x>]          [pixel#n] r
[<Param#x>]          [pixel#n] g
[<Param#x>]          [pixel#n] b
--------- reply ---------
<Status>             0(error) or 1(success)
```

11.23 (Neopixel)draw pixel

指定したLEDの色をカラーコードで設定

--------- draw pixel ---------		
<Command>	0x33	
<Param#0>	[pixel#0] x MSB	
<Param#1>	[pixel#0] LSB	
<Param#2>	[pixel#0] color code	
..		
..		
[<Param#x>]	[pixel#n] x MSB	
[<Param#x>]	[pixel#n] LSB	
[<Param#x>]	[pixel#n] color code	
--------- reply ---------		
<Status>	0(error) or 1(success)	

11.24 (Neopixel)fill rgb

全LEDのRGB色を設定

--------- fill rgb ---------		
<Command>	0x32	
<Param#0>	R	0-255
<Param#1>	G	0-255
<Param#2>	B	0-255
--------- reply ---------		
<Status>	0(error) or 1(success)	

11.25 (Neopixel)fill color

全LEDのカラーコードを設定

--------- fill color ---------		
<Command>	0x31	
<Param#0>	color code	
--------- reply ---------		
<Status>	0(error) or 1(success)	

11.26 (Neopixel)set mode

LED描画モードの設定

--------- set mode ---------			
<Command>	0x30		
<Param#0>	mode	0-3	mode for neopixel_show_task()

```
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.27 (PWM)pwm duty

PWMデューティ値の設定

```
--------- pwm duty ---------
<Command>           0x50
<Param#0>           ch                  0-3
<Param#1>           duty    MSB         (bit#24-31)
<Param#2>           duty                (bit#16-23)
<Param#3>           duty                (bit#8-15)
<Param#4>           duty    LSB         (bit#0-7)
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.28 (PWM)pwm freq

PWM周波数の設定

```
--------- pwm freq ---------
<Command>           0x51
<Param#0>           ch                  0-3
<Param#1>           freq    MSB         (bit#24-31)
<Param#2>           freq                (bit#16-23)
<Param#3>           freq                (bit#8-15)
<Param#4>           freq    LSB         (bit#0-7)
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.29 (PWM)pwm stop

PWM出力停止

```
--------- pwm stop ---------
<Command>           0x52
<Param#0>           ch                  0-3
<Param#1>           idle_level          0 or 1
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.30 (PWM)servo position

サーボデバイス用に設定されたPWM信号のサーボ位置を設定

```
--------- servo position ---------
<Command>           0x53
<Param#0>           ch                  0-3
<Param#1>           position MSB        (bit#8-15)
<Param#2>           position LSB        (bit#0-7)
--------- reply ---------
<Status>            0(error) or 1(success)
```

11.31 (SPI)spi write

SPIデバイスにデータ書き込み

```
--------- spi write ---------
<Command>           0x20
<Param#0>           port                0-1
<Param#1>           read_flag           0:ommit read_data, 1:get read_data
<Param#2>           write_data#0
[<Param#3>]         write_data#1
..
[<Param#X>]         write_data#n
--------- reply ---------      (read_flag == 0)
<Status>            0(error) or 1(success)
--------- reply ---------      (read_flag == 1)
<Status>            0(error) or 1(success)
<Data#0>            read_data#0
[<Data#1>]          read_data#1
..
[<Data#X>]          read_data#n
```

11.32 (SPI)spi read

SPIデバイスからデータ読み込み

```
--------- sqi read ---------
<Command>           0x21
<Param#0>           port      0-1
<Param#1>           read_len MSB
<Param#2>           read_len LSB
--------- reply ---------
<Status>  0(error) or 1(success)
<Data#0>            read_data#0
<Data#1>            read_data#1
..
```

<Data#X>	read_data#n

12 MGCPデバイスセキュリティについて

12.1 ネットワーク・セキュリティ

MGCP デバイスのネットワーク経由のセキュリティは、デバイスが接続している MQTT ブローカが動作しているコンピュータに依存します。MGCP デバイス自身はサーバー機能(TCP,UDP 経由の接続受け入れ機能)はデフォルトでは有効になっていませんので、MGCP デバイス自身に直接ネットワーク経由で接続される危険はありません[13]。

MGCP デバイスに対する通信は全て、MGCPデバイスが MQTTクライアントとして接続している先の MQTT ブローカを経由します。必要に応じて MQTT ブローカの認証機能やMQTT ブローカが動作している OS のアクセス制御機能を利用してください。

MGCPデバイスのコンフィギュレーションで、ログサーバーにログメッセージを送信する設定にしている場合には、UDPプロトコルを使用して直接ログサーバーに(MQTTブローカを経由しないで)メッセージを送信します。もし、ログメッセージを秘匿したい場合にはログサーバーへの送信を停止("logserver_host" コンフィギュレーション・コマンドで "127.0.0.1" を設定)してください。この場合でもMGCPデバイスのシリアルポートに直接コンソールを接続するとログメッセージは常に取得できる点に注意してください。

MGCP デバイスと Wi-Fi アクセスポイント間の通信経路を保護するために、Wi-Fi暗号化プロトコルの設定やアクセスポイント自身のパスワード管理を適切に行って下さい。また、MQTT ブローカとアクセスポイント間の通信を物理的なセキュリティの確保がしやすい LAN 内に留めるために、ローカル(会社や自宅の LAN 内)に MQTT ブローカを設置することをお勧めします。MGCP デバイスは Internet 上にあるドメイン・ホスト名や グローバルIPアドレスでアクセス可能な MQTT ブローカを接続先に設定することも可能ですが、通信経路中の盗聴や改竄の危険性があります。

設置する MGCPデバイスの MAC アドレスを予め把握しておくと、Wi-Fi アクセスポイント側の MAC アドレスプロテクト機能を併用してセキュリティを向上させることができます。MGCP デバイスのMAC アドレスは "heartbeat" イベントや abs_agent 製品の "agent_mgcp" コマンドのデバイス一覧で取得できます。

12.2 デバイス・セキュリティ

MGCP デバイスでは MQTT ブローカや Wi-Fi アクセスポイントへの接続パラメータを含む全てのコンフィギュレーション設定値は、デバイス内部の不揮発領域(NVS)に格納されています。

コンフィギュレーション・コンソールモードで MGCPデバイスを起動すると、コンフィギュレーション設定値を参照したり更新することができます。ただし Wi-Fi 設定項目については、現在の設定値を参照すると "*****" 表示になり内容を確認できないようにしています。その他の GPIO や I2C,SPI 等の設定情報は常に参照・更新することができます。

[13] UDP 経由でコマンド・リプライを送受信するダイレクトモード(dx_command)を有効にしている場合を除く

コンフィギュレーション・コマンド "password" を使用すると、コンフィギュレーション・コンソールにパスワードを設定することができます。パスワードを設定すると、コンフィギュレーション・コンソールに入るためには、正しいパスワードを入力するか、又は現在デバイスに設定されている全ての設定値をクリアした後強制的にコンソールモードに入るかを選択できるようになります。

"password" コマンドで設定したパスワードは MGCPファームウエアを上書き・更新した場合でも解除できません。

デバイスの盗難や紛失時の情報保護のために、できるだけパスワードを設定してコンフィギュレーション情報をプロテクトしておくことをお勧めます。(ただし、デモライセンスの MGCP ファームウエアでは "password" 機能は使用できません)

13 MGCPデバイス操作コマンド (abs_agent)

MGCP デバイスは MQTT ブローカ経由でコマンド文字列やリプライ、イベントを送受信しますので、MQTT クライアント機能を持っているどのようなシステムからでも操作することができます。

MGCP デバイスに対してリクエストメッセージを MQTT ブローカー経由で送信したり、その実行結果を MQTT ブローカから取得するためには、トピックの購読とリプライ処理等をクライアント側で行う必要があります。オールブルーシステムの abs_agent に同梱された agent_mgcp クライアントプログラムを使用すると、Linux シェル上から MGCP コマンドの送信とリプライ受信を簡単に実行できます。

マニュアル中のコマンド実行形式部分の記述について

● コマンド実行時には abs_agent キットをインストールしたディレクトリ中のフルパス名、またはシェルの PATH 環境変数からの相対パス名を指定します。

● コマンドパラメータが *[-r <hostname>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパラメータは省略可能です。

このドキュメントで説明している内容は、abs_agent 製品の MGCP デバイス操作に関する部分のみを抜粋しています。abs_agent 製品のセットアップ方法や他の機能については abs_agent ユーザーマニュアルをご覧ください。

13.1 agent_mgcp (MGCPデバイス管理)

- **機能説明**

  MQTT ブローカー経由で任意の MGCP デバイスにリクエストメッセージを送信した後、リプライメッセージを受信する。イベントメッセージ送信やMGCPデバイス一覧表示などの管理機能も提供する。

- **コマンド実行形式**

  (MGCP デバイス一覧を表示)

agent_mgcp *[-c] [-f <result_file>] [-r <hostname>]*

(MQTT ブローカ・エンドポイント一覧を表示)

agent_mgcp -p *[-f <result_file>] [-r <hostname>]*

(MGCPリクエストメッセージを送信)

agent_mgcp -d <mgcp_node> -m <request_json_str> *[-n] [-i <timeout>]*
[-f <result_file>] [-r <hostname>]

(MGCPイベントメッセージを送信)

agent_mgcp -d <mgcp_node> -v <event_json_str> -t <type> *[-a <application>] [-r <hostname>]*

(MGCPデバイスの現在のコンフィギュレーションをダウンロードしてファイルに保存)

agent_mgcp -d <mgcp_node> -l <config_file> *[-r <hostname>]*

(ローカルにあるコンフィギュレーションファイルをアップロードしてMGCPデバイスに設定)

agent_mgcp -d <mgcp_node> -u <config_file> *[-r <hostname>]*

(MGCPダイレクトコマンドを送信)

agent_mgcp -d <mgcp_node> -x <dxcmd_hexstr> *[-n] [-i <timeout>]*
[-f <result_file>] [-r <hostname>]

- **コマンドパラメータ**

-p

abs_agent の設定ファイルで指定した MQTT ブローカへの接続用エンドポイント一覧を表示します。 agent_mgcp コマンドでは -d<mgcp_node> で指定したノード名を元に、MQTTエンドポイントが自動的に選択されます。この処理は API ライブラリ関数 mgcp_find_endpoint() を使用しています。

ノード名変更やデバイスの入れ替えによってエンドポイントとノード名の関係が変更された場合には、-c オプションを使用して abs_agent 内部に保存・更新している変換テーブルをクリアすることができます。

-c

MGCP デバイス一覧中に表示される IP アドレスと MAC アドレス情報をクリアする。また、MGCPデバイスノード名とそのデバイスが収容されている MQTTブローカへの接続エンドポイントとの対応テーブルも同時にクリアする。

これらの情報は MGCP デバイスから定期的に送信される heartbeat イベントメッセージ中に含

まれるデータを元に abs_agent 側で保持しています。これらは常に最新の情報に更新されていますので通常はクリアする必要はありません。メンテナンス時に既存の情報を削除して最新の情報のみを参照したい場合に使用します。一旦クリアすると、MGCP デバイスから最新の heartbeat イベントを受信するまでの間は、そのデバイスへの送信はできません。

-d <mgcp_node>

リクエストメッセージやイベント送信先の MGCP デバイスのノード名を指定します。ここで指定可能なデバイスノード名とMACアドレス一覧は agent_mgcp コマンドをオプション指定無しで実行することで得られます。

"_ALL_" のノード名は全てのデバイスに対してリクエストメッセージやイベントを送信するときに常に使用できます。ただし "_ALL_" で送信されたリクエストメッセージに応答するかどうかは、各 MGCP デバイス側で設定されています。また "_ALL_" を使用した場合にはデバイスからのリプライメッセージは受信できません (-n オプションを指定したのと同等)。

16進数表記のMAC アドレスをノード名の代わりに指定することもできます。同一ノード名を持ったデバイスが複数存在する場合などに使用できます。

-m <request_json_str>

送信するリクエストメッセージを JSON 文字列で指定します。 JSON 文字列はリクエスト先の MGCP デバイスでサポートされているコマンドとオプションを指定します。 JSON 文字列中に '"' ダブルコートや "," カンマが含まれるので、JSON 文字列全体をシングルコートで囲んでパラメータを 指定してください。

-x <dxcmd_hexstr>

送信するダイレクトコマンドのコマンドとパラメータ部分のバイナリデータを 16進数文字列で指定します。MGCP デバイスから返された実行結果のステータスとデータ部分のバイナリデータを 16進数文字列ダンプします。
MGCP デバイス側でダイレクトコマンドを有効にしている場合のみ実行できます。

-n

このオプションを指定すると、MGCP デバイス側のリクエストメッセージ処理結果ステータスやリプライメッセージを受信しません。

-i <timeout>

リプライメッセージを受信するまでのタイムアウト時間(ms)を指定する。-i オプションを指定しない場合には 2000 (2秒)をデフォルト値として使用します。タイムアウト値以内にリプライメッセージを受信できなかった場合にはエラーとなります。

-v <event_json_str>

送信するイベントメッセージを JSON 文字列で指定します。JSON 文字列中に '"' ダブルコートや "," カンマが含まれるので、JSON 文字列全体をシングルコートで囲んでパラメータを指定してください。

-t <type>

イベントメッセージ送信時のイベントタイプ文字列を指定します。

-a <application>

イベントメッセージ送信時のアプリケーション文字列を指定します。このオプションを省略した場合には "abs_agent" がデフォルトのアプリケーション名として指定されます。

-l <config_file>

MGCP デバイスの現在のコンフィギュレーション情報をダウンロードしてローカルファイルに保存する。
対象となる MGCP デバイスが "Demo" ライセンスの場合にはこの機能は使用できません。

-u <config_file>

コンフィギュレーション情報を格納したファイルをアップロードして MGCP デバイスに設定する。アップロード後に、最新のコンフィギュレーションで MGCP デバイスを動作させるためには、MGCP デバイスをリセットする コマンド("reset")を実行してください。
対象となる MGCP デバイスが "Demo" ライセンスの場合にはこの機能は使用できません。

-f <result_file>

コマンド実行結果や MGCP デバイスから返されたリプライメッセージは、通常コンソールに表示されます。"-f" オプションを指定すると、これらの結果を <result_file> で指定したファイルに書き込みます。また、MQTT ブローカ・エンドポイント一覧や MGCP デバイス一覧も、同様に <result_file> に出力されます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0　　正常終了した

1　　コマンドパラメータが間違っています

2　　実行時にエラーが発生した

- **説明**

MGCP プロトコルを使用して、MQTT ブローカ経由でリクエストメッセージやイベントを送信します。

MGCP デバイス一覧は agent_mgcp コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

agent_mgcp コマンドは下記のスクリプトやイベントハンドラと連動して動作します。

スクリプト・イベントハンドラ	説明
MGCP/MGCP_EXEC	MGCP リクエストメッセージ送信とリプライメッセージ受信
MGCP/MGCP_EVENT	MGCP イベントメッセージ送信
MGCP/MGCO_NODE_CONFIG	ノードコンフィギュレーションのダウンロードとアップロード
MQTT_PUBLISH	(1) リプライメッセージ受信待ち合わせ用のイベント処理 (2) MGCP デバイスから配信される heartbeatイベントを解析して、ノード名リストと MAC アドレスリスト、IP アドレス情報を保管する。 (3) abs_agent を MGCP デバイスとして使用する場合に、abs_agent側で提供する MGCP コマンド処理

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

abs_agent が動作しているコンピュータを MGCP デバイスとして使用することもできます。この場合には MQTT_PUBLISH イベントハンドラ中の MGCP プロトコルを処理する部分に任意のリクエストコマンド処理をユーザーが組み込めます。デフォルトでは "version","hostname","echo" コマンドをサンプル実装しています。それぞれ、abs_agent のバージョン番号取得、abs_agent が動作しているコンピュータのホスト名取得、リクエストメッセージをそのまま返す機能になっています。詳しくは MQTT_PUBLISH.lua ファイルを参照してください。

また、abs_agent 側から heartbeat イベントを最低１回(できれば定期的に) "_ALL_" ノードに対して送信して、利用側のノード名リストに abs_agent のノード名を登録させてください。イベントを送信するときには mgcp_event() ライブラリ関数を使用します(イベントデータ内容は任意で空 "{}" でも構いません)。サンプルとして提供している MGCP/SEND_HEARTBEAT_EVENT スクリプトを実行すると、heartbeat イベント送信を行います。詳しくはファイルの内容を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") で動作中の abs_agent が接続している MQTT ブローカと同じ MQTT ブローカに接続している MGCP デバイス一覧を表示。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp

ServerName: raspi3     Node(s): 2
------------------------------------------------------------------------
<node_name>              <ip_address>            <MAC_address>
------------------------------------------------------------------------
M5_NO1                   192.168.100.50          B4E62D8B78C5
ESP32_NO1                192.168.100.51          240AC4077F74

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の abs_agent が接続中の MQTT ブローカー・エンドポイント一覧を表示。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -p

ServerName: raspi3     EndPoint(s): 5     Default:
------------------------------------------------------------------------------
<client_id>              <host>             <port>       <title>
------------------------------------------------------------------------------
abs9k:1111-raspi3        192.168.100.15     1883      センサーデータ登録
abs9k:444-raspi3         192.168.100.15     1883      エラー発生エンドポイント
abs9k:93501-raspi3       192.168.100.15     1883      センサーデータ送受信
abs9k:888-raspi3         192.168.100.15     1883      自宅監視用ブローカー
alarm_raspi3             192.168.100.15     1883      アラーム装置MQTT接続

pi@raspi3:~/abs_agent$
```

● MGCP デバイス・ノード名 "ESP32_NO1" の GPIO#27 を入出力モードに設定した後、ポート値を High に設定する。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_mode","gpio":27,"mode":"in_out"}'
{"result":"success","error_text":""}
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_set","gpio":27,"level":1}'
{"result":"success","error_text":""}
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_get","gpio":27}'
```

```
{"result":"success","error_text":"","level":1}
pi@raspi3:~/abs_agent$
```

● 全MGCP デバイスにイベントタイプ "test_message" でイベント内容が {"message":"hello,world"} の JSON 文字列を送信する。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -d _ALL_ -t test_message -v '{"message":"hello,world"}'
pi@raspi3:~/abs_agent$
```

● 全MGCP デバイスにリセットコマンドを送信して全てのデバイスを強制的に再起動させる。

```
pi@mars:~/abs_agent$ ./agent_mgcp -d _ALL_ -n -m '{"command":"reset"}'
pi@mars:~/abs_agent$
```

14 MGCPデバイス操作ライブラリ関数 (abs_agent)

MGCP デバイスは MQTT ブローカ経由でコマンド文字列やリプライ、イベントを送受信しますので、MQTT クライアント機能を持っているどのようなシステムからでも操作することができます。

MGCP デバイスに対してリクエストメッセージを MQTT ブローカー経由で送信したり、その実行結果を MQTT ブローカから取得するためには、トピックの購読とリプライ処理等をクライアント側で行う必要があります。オールブルーシステムの abs_agent を使用するとこれらの処理をライブラリ関数として簡単に使用することができます。

ここでは、abs_agent の Lua ユーザースクリプト(イベントハンドラ)で使用可能な MGCP デバイス操作用ライブラリ関数について説明します。

ライブラリ関数定義のパラメータ記述で *[,<modulename>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパラメータは省略可能です。複数のパラメータを指定するときに、途中のパラメータのみを省略指定する場合には、パラメータ値に nil を指定してください。

このドキュメントで説明している内容は、abs_agent 製品の MGCP デバイス操作に関する部分のみを提供しています。abs_agent 製品のセットアップ方法や詳しい機能については abs_agent ユーザーマニュアルをご覧ください。

14.1 mgcp_command()

- **機能概要**

  MGCP プロトコルを使用したリクエストコマンド(JSON)を MQTT ブローカ経由でデバイスに送信した後、リプライ(JSON)を受信する。

- **関数定義**

stat [,reply] = mgcp_command(node, request [,noreply [,timeout]])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。 パラメータ指定が間違っている場合や、リプライ受信時にタイムアウトした場合には false が返ります。 指定したサービスが提供されていない等で、デバイス側のコマンド処理が失敗した場合でも、リプライメッセージを受信した時点で true が返る事に注意してください。デバイス側のコマンド処理が正常に終了したかどうかは、reply で返される JSON 文字列中の "result" タグの値が "success" になっているかで判断できます。
reply:String	リクエスト先から返されたリプライ JSON 文字列
node:String	リクエストコマンド送信対象デバイスに設定したノード名 対象となるデバイスがサポートしている場合には、全ノードを対象として送信する場合の "_ALL_" や、リクエスト送信先を厳密に指定するためMAC アドレス(16進数文字列、例:"240AC4077F74"など)を指定することができます。
request:String	リクエストコマンドを格納した JSON文字列。 ここで指定するコマンドは、送信対象ノード側で提供しているコマンドを指定すると。
noreply:boolean	リプライを受信しないことを指定する。true または false を指定。 パラメータ省略時は false でリプライ受信まで内部で wait する。
timeout:Number	リプライ受信までのタイムアウト(ms) を指定する。 パラメータ省略時は 2000ms(2秒) が指定されます。

- **備考**

node パラメータに指定したノード名がどの MQTT ブローカ(MQTT エンドポイント)に属しているかは、ライブラリ関数内部で mgcp_find_endpoint() をコールすることで自動的に選択されます。

node に "_ALL_" を指定した場合には、全ての MQTT エンドポイントに対して同じリクエストメッセージが送信されます。

リプライ受信用の処理は、このライブラリ関数を実行するコンピュータ上で、abs_agent の MQTT_PUBLISH イベントハンドラ中に下記のステートメントを予め記述しておく必要があります。このスクリプトはインストール時のデフォルト・イベントハンドラ(scripts/MQTT_PUBLISH.lua)中に記述されています。もしコメントアウトされている場合には、ライブラリ関数を使用する前に有効にしてください。(下記参照)

```
local topic_arr = topic_to_tbl(g_params["PublishTopic"])
if #topic_arr >= 3 then
        local msg_id = topic_arr[4]
        ------------------------------------------------------------------------------------------------------------
        -- このコンピュータからリクエストコマンドが送信された場合には、MQTT ブローカ経由でサーバーから送り返された
```

```
        -- リプライメッセージをここで処理する。
        -- リプライメッセージのトピック名は /reply/<リクエスト元ホスト名(このコンピュータ名)>/<リクエスト先ノード名>/<msg_id>
        -- で、<msg_id> はリクエスト時に送信した <msg_id>と同じ文字列が使用されている。
        -- リクエスト時に使用した <msg_id> の名前のグローバル共有変数を、このコンピュータからリクエスト
        -- を行ったかどうかを判断するフラグとして使用する。また同時に、複数のサーバーからリプライを受信した場合に
        -- 最初に受信したリプライのみを選択するためのフラグとしても利用する。
        -- MQTT ブローカ経由で受信したサーバーのリプライメッセージは、アンダースコア文字を追加した "_<msg_id>" の名前の
        -- グローバル共有変数にセットする。
        -- その後、イベントをセット状態にして、リクエスト発行側で待ち状態になっているタスクを復帰させる。
        ------------------------------------------------------------------------------------------------------------------
        if (topic_arr[1] == "reply") and (msg_id ~= nil) then
          local stat,val = get_shared_data(msg_id,true)
          if val == "1" then
            set_shared_data("_" .. msg_id,PublishString,true)
            event_set2(msg_id)
          end
        end -- reply 処理終了
end
```

(scripts/MQTT_PUBLISH.lua ファイルの MGCP リプライメッセージ受信処理を行う部分)

このライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/MGCP_LIB.lua ファイルが自動的にロードされて定義されます。詳しい MGCP プロトコルの仕様については、上記ファイル中のコメント欄を参照してください。

- **使用例**

```
local stat,reply = mgcp_command("自宅監視用ブローカー","ESP32Node1",'{"command":"version"}',false,2000)
if not stat then error() end
if reply then script_result(g_taskid,"Reply",reply) end
```

14.2 mgcp_event()

- **機能概要**

  MGCP プロトコルを使用してイベント(JSON)を MQTT ブローカ経由でデバイスに送信する。

- **関数定義**

  stat = mgcp_event(node, type, event *[,application]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
  　　パラメータ指定が間違っている場合には false が返ります。

  node:String　　イベント送信対象のノード名

対象となるデバイスがサポートしている場合には、全ノードを対象として送信する場合の "_ALL_" や、リクエスト送信先を厳密に指定するたのMAC アドレス(16進数文字列、例:"240AC4077F74"など)を指定することができる。

type:String　　イベント送信時のトピック文字列中の <event_type_string>を指定する

event:String　　イベントを格納した JSON文字列

application:String　　イベント送信時のトピック文字列中の <application_name>を指定する。省略時は "abs_agent" が使用される。

- **備考**

node パラメータに指定したノード名がどの MQTT ブローカ(MQTT エンドポイント)に属しているかは、ライブラリ関数内部で mgcp_find_endpoint() をコールすることで自動的に選択されます。

node に "_ALL_" を指定した場合には、全ての MQTT エンドポイントに対して同じイベントメッセージが送信されます。

このライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/MGCP_LIB.lua ファイルが自動的にロードされて定義されます。詳しい MGCP プロトコルの仕様については、上記ファイル中のコメント欄を参照してください。

- **使用例**

```
local stat = mgcp_event("_ALL_","gpio_change_detect",'{"gpio":23,"direction":"rise"}')
if not stat then error() end
```

14.3　mgcp_find_endpoint()

- **機能概要**

MGCP デバイスのノード名から MQTT エンドポイント名を取得する。

- **関数定義**

stat *[*,title_or_id*]* = mgcp_find_endpoint(node)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
パラメータ指定が間違っている場合や、エンドポイントが見つからなかった場合には false が返ります。
MQTT エンドポイントは、MGCP デバイスが定期的に送信している heartbeat イベント受信時に保存しているノード名とエンドポイントの対応表を元に検索します。複数のエンドポイントから受信した heartbeat イベントで、同一のノード名が存在している場合には、最初に見つけたエンドポイントを返します。

title_or_id:String　　MQTT サービスに登録済みのエンドポイントタイトル名または ClientID

node:String　　検索対象の MGCP デバイスに設定したノード名

- **備考**

title_or_id で取得される MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録されているものを使用します。Title が登録されている場合には ClientID よりも優先されて取得します。

このライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/MGCP_LIB.lua ファイルが自動的にロードされて定義されます。詳しい MGCP プロトコルの仕様については、上記ファイル中のコメント欄を参照してください。

14.4　mgcp_lcd_print()

- **機能概要**

MGCP デバイス(M5StickC/M5Stack) の LCD に ASCII文字列を表示する。

このライブラリ関数で表示できるのは英数字のみです。日本語を含む文字列を表示する場合には mgcp_lcd_jprint(), mgcp_lcd_jprintln() を代わりに使用してください。

- **関数定義**

stat = mgcp_lcd_print(node,str*[,x0][,y0][,wrap][,cls][,font][,fg][,bg]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。 パラメータ指定が間違っている場合や、エンドポイントが見つからなかった場合には false が返ります。
node:String	MGCP ノード名
str:string	表示する ASCII文字列
x0:number(option)	表示開始座標(X軸) default: wrap が trueの場合 0,false の場合は str を画面中央に表示する座標
y0:number(option)	表示開始座標(Y軸) default: wrap が trueの場合 0,false の場合は str を画面中央に表示する座標
wrap:boolean(option)	長い文字列を自動的に折り返す default: false 折り返す場合に、縦方向のフォントサイズの 1/5 に相当するピクセル分を空ける。
cls:boolean(option)	文字列描画前に画面をクリアする default: false
font:number(option)	フォント番号 0..9 の整数を指定 default: 以前に設定した値またはデバイス初期値の 2
fg:number(option)	文字色のカラーコードを指定する　　default:15 (WHITE)
bg:number(option)	背景色のカラーコードを指定する　　default:0 (BLACK)

- **備考**

このライブラリ関数は abs_agent 起動時に scripts/preload/200_MGCP/002_DEVICE/M5_STACK_LCD.lua ファイルが自動的にロードされて定義されます。詳しい仕様については、上記ファイル中の Luaソースコードを参照してください。

14.5 mgcp_lcd_jprint()

- **機能概要**

  MGCP デバイス(M5StickC/M5Stack) の LCD に日本語文字列を表示する。

- **関数定義**

  stat = mgcp_lcd_jprint(node,str*[,x0][,y0][,wrap][,cls][,font][,scr_w][,scr_h][,split][,size][,fg][,bg]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。 パラメータ指定が間違っている場合や、エンドポイントが見つからなかった場合には false が返ります。	
node:String	MGCP ノード名	
str:string	日本語文字列。ASCII 文字は同じ字形の全角文字に自動変換される。 各文字のフォントデータが見つからなかった場合にはその文字は表示しない。	
x0:number(option)	表示開始座標(X軸) default: 0	
y0:number(option)	表示開始座標(Y軸) default: 0	
wrap:boolean(option)	長い文字列を自動的に折り返す default: false 折り返す場合に、縦方向のフォントサイズの 1/5 に相当するピクセル分を空ける。	
cls:boolean(option)	文字列描画前に画面をクリアする default: false	
font:string(option)	グローバル共有エリアにロード済みの日本語フォントのチャンネル名 default: "SHINONOME_FONTX"	
scr_w:number(option)	LCD 表示器の横サイズを指定。 default:デバイスタイプによって自動設定	
scr_h:number(option)	LCD 表示器の縦サイズ default:デバイスタイプによって自動設定	
split:number(option)	指定した文字数ごとに、複数の MGCP コマンドに分割して文字を表示する default:nil パラメータ省略時は全ての文字を一度のコマンドで表示する。ただしこの場合 MGCP コマンドのリクエスト文字列が 4096bytesを超える場合にエラーが発生する。	
size:number(option)	文字を size 倍に拡大して表示する	default:1
fg:number(option)	文字色のカラーコードを指定する	default:15 (WHITE)
bg:number(option)	背景色のカラーコードを指定する	default:0 (BLACK)

- **更新するグローバル共有変数**

LCD に最後に表示した"文字"の次の表示開始予定座標をグローバル共有変数に保存する。

前回表示した文字列に続けて表示したい場合に、次に表示開始座標を計算するときに利用できる。

mgcp_lcd_jprintln() ライブラリ関数の内部でこのグローバル変数を利用している。

画面のサイズは考慮していないので折り返し表示が必要な場合は座標値の再計算が必要。

キー名	値
MGCP_LCD_PREV_<node>	表示した最後の文字の情報 (CSV形式で <x0>,<y0>,<fh>,<size>) <x0>: 表示した最後の文字の次の開始座標 X軸 <y0>: 表示した最後の文字の次の開始座標 Y軸 <fh>: 表示した最後の文字の縦方向のフォントサイズ <size>: 表示した最後の文字のフォント拡大倍率(size パラメータに指定したもの)

- **備考**

  このライブラリ関数は abs_agent 起動時に scripts/preload/200_MGCP/002_DEVICE/M5_STACK_LCD.lua ファイルが自動的にロードされて定義されます。詳しい仕様については、上記ファイル中の Luaソースコードを参照してください。

14.6 mgcp_lcd_jprintln()

- **機能概要**

  MGCP デバイス(M5StickC/M5Stack) の LCD に日本語文字列を表示する。

  前回 LCD に表示した文字列の終端で改行して画面左端から表示して、長い文字列は常に自動的に折り返して表示する。

- **関数定義**

  stat = mgcp_lcd_jprintln(node,str*[,cls][,font][,scr_w][,scr_h]*
  [,split][,size][,fg][,bg])

- **パラメータとリターン値**

node:String	MGCP ノード名
str:string	日本語文字列。ASCII 文字は同じ字形の全角文字に自動変換される。 各文字のフォントデータが見つからなかった場合にはその文字は表示しない。
cls:boolean(option)	文字列描画前に画面をクリアする default: false また、true 指定時は文字の表示開始位置を画面左上に戻す。
font:string(option)	グローバル共有エリアにロード済みの日本語フォントのチャンネル名 default: "SHINONOME_FONTX"
scr_w:number(option)	LCD 表示器の横サイズ default:デバイスタイプによって自動設定
scr_h:number(option)	LCD 表示器の縦サイズ default:デバイスタイプによって自動設定
split:number(option)	指定した文字数ごとに、複数の MGCP コマンドに分割して文字を表示する

default:nil パラメータ省略時は全ての文字を一度のコマンドで表示する。ただしこの場合 MGCP コマンドのリクエスト文字列が 4096bytesを超える場合にエラーが発生する。

size:number(option)	文字を size 倍に拡大して表示する	default:1
fg:number(option)	文字色のカラーコードを指定する	default:15 (WHITE)
bg:number(option)	背景色のカラーコードを指定する	default:0 (BLACK)

- **備考**

このライブラリ関数は abs_agent 起動時に scripts/preload/200_MGCP/002_DEVICE/M5_STACK_LCD.lua ファイルが自動的にロードされて定義されます。詳しい仕様については、上記ファイル中の Luaソースコードを参照してください。

14.7 mgcp_dx_command()

- **機能概要**

ダイレクトコマンドをUDP データグラムソケットで送信する。

- **関数定義**

stat *[,*rpl_arr*]* = mgcp_dx_command(node, req_arr *[,*noreply *[,*timeout*]]*)

- **パラメータとリターン値**

stat:Boolean 成功した場合は true, 失敗した場合は false が返る。
パラメータ指定が間違っている場合や、リプライ受信時にタイムアウトした場合には false が返ります。
指定したサービスが提供されていない等で、デバイス側のコマンド処理が失敗した場合でも、リプライメッセージを受信した時点で true が返る事に注意してください。コマンド処理が正常に終了したかどうかは、reply_arr で返されるデータ中の の Status バイトの値が 1 になっているかで判断できます。

rpl_arr:Table [1..#max_item] of Number
リクエスト先から返されたステータスとデータが格納されたデータ配列

node:String リクエストコマンド送信対象デバイスのノード名

req_arr:Table [1..#max_item] of Number
コマンドとパラメータが格納されたデータ配列

noreply:boolean リプライを受信しないことを指定する。true または false を指定。
パラメータ省略時は false でリプライ受信まで内部で wait する。

timeout:Number リプライ受信までのタイムアウト(ms) を指定する。
パラメータ省略時は 1000ms(1秒) が指定されます。

- **備考**

mgcp_dx_command() ライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/DXCMD_LIB.lua

ファイルが自動的にロードされて定義されます。詳細はスクリプトファイルを参照して下さい。

14.8 dxcmd_exec_stat_check()

- **機能概要**

  ダイレクトコマンドのリクエストフレームを送信して結果ステータスをチェックする

  正常にリプライを受信した後、リプライデータ中のステータスバイトが 0x01 の時のみ

  true を返しそれ以外は false を返す

- **関数定義**

  stat = dxcmd_exec_stat_check(node, frame)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  パラメータ指定が間違っている場合や、リプライ受信時にタイムアウトした場合には false が返ります。

  frame:Table [1..#max_item] of Number

  コマンドとパラメータが格納されたデータ配列

  node:String　　リクエストコマンド送信対象デバイスのノード名

- **備考**

  dxcmd_xxxxx() ライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/DXCMD_LIB.lua

  ファイルが自動的にロードされて定義されます。詳細はスクリプトファイルを参照して下さい。

14.9 dxcmd_exec_reply_get()

- **機能概要**

  ダイレクトコマンドのリクエストフレームを送信して結果ステータスをチェックする

  正常にリプライを受信した後、リプライデータ中のステータスバイトが 0x01 の時のみ

  trueとリプライフレーム中のデータ部分を返しそれ以外は false を返す

- **関数定義**

  stat *[*,reply*]* = dxcmd_exec_reply_get(node, frame)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  パラメータ指定が間違っている場合や、リプライ受信時にタイムアウトした場合には false が返ります。

  reply:Table [1..#max_item] of Number

  ステータスとリターンデータが格納されたデータ配列

  frame:Table [1..#max_item] of Number

コマンドとパラメータが格納されたデータ配列

node:String　　リクエストコマンド送信対象デバイスのノード名

- **備考**

dxcmd_xxxxx() ライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/DXCMD_LIB.lua ファイルが自動的にロードされて定義されます。詳細はスクリプトファイルを参照して下さい。

14.10 dxcmd_param_add_byte(),dxcmd_param_add_word(), dxcmd_param_add_long()

- **機能概要**

ダイレクトコマンドのリクエストフレームにバイトデータ、ワードデータ、ロングワードデータを追加する。

- **関数定義**

dxcmd_param_add_byte(frame, b)

dxcmd_param_add_word(frame, w)

dxcmd_param_add_long(frame, l)

- **パラメータとリターン値**

frame:Table [1..#max_item] of Number

コマンドとパラメータが格納されたデータ配列

b:Number　　フレームに追加するデータ。下位1バイトを取り出してframe 配列に追加する。

w:Number　　フレームに追加するデータ。下位2バイトを取り出してframe 配列に追加する。

このとき上位バイトから順にframe に追加する(BigEndian形式)。

l:Number　　フレームに追加するデータ。4バイト分を取り出してframe 配列に追加する。

このとき上位バイトから順にframe に追加する(BigEndian形式)。

- **備考**

dxcmd_xxxxx() ライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/DXCMD_LIB.lua ファイルが自動的にロードされて定義されます。詳細はスクリプトファイルを参照して下さい。

14.11 dxcmd_param_append()

- **機能概要**

ダイレクトコマンドのリクエストフレームに別の配列データを追加する。

- **関数定義**

dxcmd_param_append(frame,tbl)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

パラメータ指定が間違っている場合や、リプライ受信時にタイムアウトした場合には false が返ります。

frame:Table [1..#max_item] of Number

コマンドとパラメータが格納されたデータ配列

tbl:Table [1..#max_item] of Number

追加するバイナリデータが格納されたデータ配列

- **備考**

  dxcmd_xxxxx() ライブラリ関数は abs_agent 起動時に scripts/preload/012_MQTT_MGCP/DXCMD_LIB.lua ファイルが自動的にロードされて定義されます。詳細はスクリプトファイルを参照して下さい。

15 本製品に使用したソフトウェアライセンス表記

15.1 FastLED

URL: https://github.com/FastLED/FastLED

The MIT License (MIT)

Copyright (c) 2013 FastLED

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

15.2 ESP-IDF component to work with M5StickC

URL: https://github.com/pablobacho/m5stickc-idf

16　ライセンス・サポート

オールブルーシステムは、正規ライセンスを取得されたお客様の所有するコンピュータ(ESP32プロセッサ搭載のコンピュータや無線モジュール)に本ソフトウエアをインストールし使用する権利を提供します。

お客様は、本ソフトウェアを複製・複写することや、これに対する修正・追加等の改変をすることはできません。

abs_agent とセットの MGCPライセンスを購入された場合は、abs_agent のライセンスを取得したコンピュータ(Raspberry Piや PC/AT互換機)を設置した場所と同一施設内(事務所、工場、家屋など)に MGCPデバイスを設置する場合に限り、必要な個数のコンピュータに本ソフトウエアをインストールすることができます。このとき、abs_agentが動作していない状態であっても、同一施設内であれば任意の数の MGCPデバイスを使用できます。１つのシステムとして複数の設置場所に分散設置した MGCP デバイスを使用・運用する場合に、それぞれの設置場所が同一施設内でない場合(住所が異なっている場合)には、それぞれの設置場所の施設毎にライセンスが必要となります。

オールブルーシステムから発行したライセンスを、ライセンスに記載されたお客様とコンピュータ以外に ″オールブルーシステム製品″ として再ライセンスを行う事や、貸与又はリースその他の方法で第三者に使用させることはできません。

MGCP ライセンスをご購入になる前に、目的の機能がお客様の環境で動作するかどうかを確認されることをお勧めします。オールブルーシステムの MGCP プログラムは、全ての機能を事前にお客様が検証可能なように、デモ用のファームウエアをオールブルーシステムのホームページ上で提供しています。

動作環境を満たしている場合でも、お客様の環境やハードウエアによっては正常に動作しない場合があります。最善のサポートをご提供致しますが、こちらで再現できない問題につきましては十分なサポートができない場合がありますす、事前にご了承下さい。

お客様独自のライセンス形態を希望される場合でも対応できます。また、ファームウエアのカストマイズやオールブルーシステムでのファームウエアの書き込み作業も可能ですので、遠慮なくお問い合わせください。
(問い合わせ先メールアドレス contact@allbluesystem.com)

17 更新履歴

REV A.1.4 2021/9/28

ダイレクトコマンドの説明を追加

"i2c_config", "spi_config", "dx_config", "job_config", "job_once" MGCP コマンド追加

"nvs_get", "nvs_set" MGCP コマンドに "blob"タイプ追加

REV A.1.3 2021/3/23

PWM/SERVO 信号出力機能追加

REV A.1.1 2020/6/17

ログサービス機能について記述追加

"force_config" MGCPコマンド追加

REV A.1.0 2020/5/11

初版作成