UECS ノード開発用ミドルウェア

UARDECS_ESP32C3 Ver.1.0.0 要点説明書

著者: 星　岳彦

2026 年 10 月 14 日版

目　次

1.はじめに 2
2.対応基板と ESP32C3 を使用する際の注意点 3
　2.1 セーフモードピンと WiFi 接続の注意点 3
　2.2 CPU 基板の多目的ポート(GPIO)使用に関する注意点 4
　2.3 ESP32C3 の CPU を使用する場合の注意点 5
　2.4 ESP32C3 版で使用する際の注意点 5
3.ArduinoIDE での UARDECS_ESP32C3 の設定方法 6
4.ESP32C3 版のセーフモードとノーマルモードの注意点 7
5.サンプルプログラムによる使用法の習得 7
6. UARDECS_ESP32C3 の変数および関数の仕様(参照用) 8
　6.1 定義が必要なグローバル定数と変数(型・変数名・内容) 8
　6.2 スケッチで実装が必要な関数(関数名・詳細) 10
　6.3 スケッチで必要に応じて呼び出せる関数 12
7.おわりに 12

1.はじめに

UARDECS (ユアルデックス)ESP32C3 版は開発環境に無償で入手可能な Arduino IDE を用い小型低コスト CPU である ESP32C3 を実装した基板を利用してユビキタス環境制御システム(UECS、https://uecs.jp)のノードを作成するために開発されたミドルウェアです。 UECS においては、UECS-CCM(共用通信子)と命名されたオープン通信プラットフォームで、お互いに通信することで情報共有と連携を図ります。本ミドルウェアは UECS-CCM に準拠した通信を実装しており、UECS 準拠のノード開発を容易にします。UECS-CCM は、通信規約文書(https://www.uecs.jp/outline/kiyaku)が UECS 研究会からダウンロードできますので、ご参照ください。

ESP32C3 は、I/O 点数が比較的少ないものの、WiFi や Bluetooth が搭載されていて、しかも数 cm 角の基板で IoT を実現できて超小型です。基板コストも数百円から千円程度で、センサやアクチュエータごとに内蔵して、当初の理念であった真の自律分散制御の達成を可能にします。本ミドルウェアを使用することで、Arduino、Raspberry PICO で Arduino IDE で開発された UECS ノードのソフトウェア資産を容易に移植可能にしました。また、UECS-CCM(共用通信子)の搭載通信プロトコルに関しては、UARDECS_MEGA 版と同様の仕様になっています。

Github サイト(https://github.com/uecs-star/UARDECS_ESP32C3)にて、本ミドルウェアはオープンソースで公開されています。また、ArduinoIDE のライブラリ管理から、「UARDECS_ESP32C3」と検索してインストール可能になっています。本ライブラリは以下のライセンスで提供されます。

『

Arduino IDE 用ライブラリソフトウェア「UARDECS_ESP32C3」は、
2 条項 BSD ライセ ンスの下で配布されています。

Copyright 2026 Takehiko Hoshi, Hideto Kurosaki and Ken-ichiro Yasuba,
Wakayama, Japan

ソースコード形式およびバイナリ形式での再配布および使用は、
変更の有無に かかわらず、以下の条件が満たされる場合に許可されます。

ソースコードを再配布する場合、上記の著作権表示、この条件のリスト、
および以下の免責事項を保持する必要があります。

バイナリ形式で再配布する場合、上記の著作権表示、この条件のリスト、
および以下の免責事項を、配布物と共に提供されるドキュメントやその他の
資料に記載する必要があります。

本ソフトウェアは、著作権者および貢献者によって「現状のまま」提供され、 商品性および特定目的への適合性に関する黙示の保証を含む（ただしこれらに 限定されない）いかなる明示的または黙示的な保証も否認されます。いかなる 場合においても、著作権者または貢献者は、本ソフトウェアの使用から生じる 直接的、間接的、付随的、特別、懲罰的、または派生的損害（代替品または代 替サービスの調達、使用・データ・利益の損失、または業務の中断を含むがこ れらに限定されない）について、たとえそのような損害の可能性について知ら されていたとしても、契約、無過失責任、または不法行為（過失その他を含む） のいずれの責任理論に基づいても、一切の責任を負いません。

』

本ミドルウェアは、著者らにより GPT Codex 6.1 Sol の支援を受けて、UARDECS_PICO 版をひな形に開発されました。本ミドルウェアの起源は、「低コスト環境制御システム構築のためのプログラムライブラリの開発(安場健一郎ほか、2016 年)」・農業情報研究 25 巻 1 号 p. 12-18 (https://doi.org/10.3173/air.25.12)です。

2.対応基板と ESP32C3 を使用する際の注意点

本ミドルウェアは、Arduino IDE2.3.10 で確認済みです。CPU 基板は、0．42インチの表示器を装着した ESP32C3-OLED、XIAO ESP32C3 で動作確認しています。これらの互換品、および、ESP32C3 搭載の各種基板について、ArduinoIDE で適切なボードを選ぶことで動作可能と思います。ただし、無保証ですので、自己責任でお使いください。検証済みの 2 種の基板を使用する場合の注意点は以下の通りです。

2.1 セーフモードピンと WiFi 接続の注意点

(a)セーフモード(最初に DHCP 接続するときのモード)は GPIO 3 の端子の接続で判定します。XIAO ESP32C3 では基板上の D1 が GPIO 3 です（基板上の D3 は GPIO 5 なので注意）。D1 を GND(0V)に接続するとセーフモードになります。開放するとノーマルモード(Web 画面で設定した IP アドレスで接続する)になります。

(b)ESP32C3 のネットワークは WiFi が標準装備されていて、それを使います。このため、setup()内の setWiFiCredentials("SSID", "Password");に、接続可能な WiFi ネットワークの SSID と Password を記載してください。セーフモードでは、この接続情報で DHCP(自動アドレス設定)により接続されます。

(c)ノーマルモードでの接続は、Web 画面の Network Config で設定した、SSID、IP アドレス等の情報になります。セーフモードでこれらの情報を正しく設定してから、GPIO 3(XIAO では D1)端子を開放してください。

(d)内蔵されている WiFi 通信チップにあらかじめ MAC(物理)アドレスが設定されていますので、以前の UARDECS で必須だった設定は不要になりました。MAC3 アドレスを知りたい場合は、WiFi.macAddress()で返されます。

2.2 CPU 基板の多目的ポート(GPIO)使用に関する注意点

***ESP32C3+0.42inchOLED

・GPIO2：起動モードを設定するストラッピングピンの 1 つです。電源が入る瞬間にこのピンの電位が変化するような回路（強力なプルダウンなど）を接続していると、正常に起動しなくなる場合があります。起動後は通常の GPIO として使用可能です。

・GPIO5：OLED の SDA（データ線）に接続

・GPIO6：OLED の SCL（クロック線）に接続

・GPIO8：オンボードの青色 LED に接続

・GPIO9：BOO0（ブート）ボタンに接続※起動モードを決定する重要なピンのため、外部回路で常に GND へ落とすような使い方をするとプログラムの書き込みや起動ができなくなります。

・GPIO12 ～ GPIO17：内蔵 SPI フラッシュメモリ専用プログラムを読み出すために常に高速通信を行っているため、汎用 GPIO としては一切使用できません。

・GPIO18, GPIO19：USB-JTAG / USB シリアル変換機能 Type-C ポートからのプログラム書き込みやシリアルモニター通信で使用されています。プログラム上で設定を無効化すれば GPIO として使えないこともないですが、PC との通信ができなくなるリスクがあるため避けるのが無難です。

***XIAO ESP32C3

・D2（GPIO2）理由: こちらも起動モードに関係するストラッピングピンです。プルアップやプルダウンが内部でされていないため、接続する機器によっては起動失敗の原因になります。

・D3（GPIO5 / A3 など）: 一部の環境（Arduino IDE の古い定義など）でアナログ入力（A3）として正常に認識・定義されていないケースや、Wi-Fi/BLE 通信時にアナログ値の読み取りに影響を受ける場合があります。また JTAG 信号線（MTDI など）と兼用です。

・D4,D5（GPIO6,7）は I2C(Wire)の SDA と SCL に使用するので、別用途には使用しない方が無難です。

・D6 は起動時に UART 出力に設定されるので、もし D6 を入力として使うと不用意に大電流が流れてしまう危険性があります。出力で使うようにしましょう。

・D8（GPIO8）: 起動モード（Boot Mode）を決定するストラッピングピン（Strapping Pin）の一つです。起動時に LOW かつ GPIO9 も LOW になると予期せぬ挙動や特殊モード（UART_BOOT）に入るため、外部回路を接続する際は起動時の電位（HIGH に維持する等）に注意が必要です。
・D9（GPIO9）: オンボードの BOOT ボタン とプルアップ抵抗に接続されています。スイッチ入力としては最適ですが、起動時に GND に落ちるような回路や、強い外部負荷を接続すると正常に起動（ダウンロードモード）できなくなります。

また、電圧が 3.3V であること、ピンに流せる電流は、GPIO ピン 1 本あたりに流せる電流は、公式データシートによると、ドライブ能力を最大（PAD_DRIVER = 3）に設定した場合、ソース電流（出力・吐き出し）が最大 40mA、シンク電流（入力・吸い込み）が最大 28mA です。

2.3 ESP32C3 の CPU を使用する場合の注意点

(1)セーフモードにする信号ピンは GPIO 3 に設定しています。XIAO ESP32C3 ボードの場合は D1 ピンになります。これを GND(0V)にすることでセーフモードになります。ピンの状態チェックは UARDECS のセットアップ時に一度だけです。以降変えても影響ありません。
(2)ESP32 から const 宣言だけでフラッシュメモリに割り当てられるので PROGMEM=を使う必要はなくなりました。
(3)ESP32C3 にはあらかじめ mac アドレスが設定されていますので、定義不要になりました。また、WiFi 接続になりますのでもセーフモードの時の SSID と password の設定が必要です。メモリ容量節約のため文字長は 25 文字までにしています。(25 文字以上は設定できません)そして、セーフモードでは DHCP(自動アドレス)接続になり、自動でアドレス設定されます。

2.4 ESP32C3 版で使用する際の注意点

(1)Ethernet 接続から WiFi 接続になり、それに伴った仕様変更が実施されています。WiFi 特有のセキュリティー設定(WiFi 端末分離機能など)でブロードキャスト CCM が他のノードで受信できない現象の発生が報告されています。WiFi ブリッジ等のセキュリティー設定を見直し、ネットワークアナライザ等で配送確認してください。
(2)UDP16521 ポートを使う提供者サーチ CCM の TYPE 属性に対する、提供者応答の挙動が変わりました。提供者を検索する TYPE 属性の検索キーは、これまでは完全一致した提供者だけが応答していました。ESP32C3 版では、部分一致した提供者も応答するようにしました。CCM 識別子命名規約の項目名やノード種別名が同一の提供者を容易にサーチ可能です。ただ、このプロトコルを使用するアプリケーションでは、より多数の提供者からの応答 CCM が送信されますので、バッファオーバーフローに注意してください。

(3)本ライブラリは Arduino に搭載されていた EEPROM の代用として ESP32C3 のプログラムフラッシュの未使用領域 4096byte を保存領域に使用します。また、SRAM を 4096byte 分キャッシュ領域として使用する(起動後に確保されるので注意)ので、この分だけ利用可能なメモリが減少します。

EEPROM の耐久性は書き換え回数のみは Arduino と同等（100,000 回)ですが、4096byte 単位でしか書き換えができないため、一部書き換えただけでも 4096byte の領域全体の耐久性を損耗します。これらは書き換え頻度が少ない IP アドレスの保存やその他設定データの保存には深刻な影響はないかもしれません。しかし、データのロギングなど一定時間間隔で書き込むような用途には著しく不利になります。(おそらく)byte 単位で書き換えが可能な Arduino より遥かに少ない回数で限界に達します。プログラム中で EEPROM を高頻度に書き換える場合、別途不揮発性のデータが保存可能なデバイスを接続する設計にすることを推奨します。

3.ArduinoIDE での UARDECS_ESP32C3 の設定方法

(a)ESP32 用ボードライブラリのインストール

メニューの [ファイル] ＞ [環境設定] を開きます。「追加のボードマネージャーの URL」に次の URL を入力して [OK] を押します。https://githubusercontent.com

(b)ESP32 ボードのインストール

メニューの [ツール] ＞ [ボード] ＞ [ボードマネージャー] を開きます。検索窓に esp32 と入力します。Espressif Systems の「esp32」をインストールします。

(c)パソコンと ESP32-C3 ボードを USB ケーブルで接続します。

メニューの [ツール] ＞ [ボード] ＞ [ESP32 Arduino] から使用しているボード（例: ESP32C3 Dev Module や XIAO_ESP32C3 など）を選択します。接続した [シリアルポート] を選択します

(d) USB CDC On Boot を Enabled にする

USB 端子が ESP32-C3 の GPIO18/19 へ直結された基板（XIAO や多くの 0.42 インチ OLED 基板）は USB CDC On Boot を Enabled にしないと、コンパイル・書き込みが終了後の実行時に Serial.print()等でコンソールに文字表示できません。メニューの [ツール] ＞ [USB CDC On Boot: Enabled]にします。ただし、CP210x/CH340 等の USB-UART 変換 IC を搭載した基板は Disabled にします。

以上の処理でコンパイル可能な環境が整います。(c) (d)の操作は同じ型番でもボードを交換するたびに実行してください。

(e)IDE 画面左上の→マークをクリックするとコンパイルとプログラムの転送が始まります。
(f)転送が終わると自動的にボードがリセットされプログラムが実行されます。

4.ESP32C3 版のセーフモードとノーマルモードの注意点

UARDECS には、プログラムの再書き込み無しで、Web 設定画面からノードを適切に IP 接続するためのセーフモードが用意されています。セーフモードは GPIO3 または、D1 端子を GND(0V)に接続して起動すると有効化します。通信に WiFi を使用しているため、以下の通り、従来版 UARDECS とはセーフモードの仕様が異なります。

セーフモードでは、setup()関数の中で、UECSsetup();の実行前に、setWiFiCredentials(SSID, PASS);を必ず実行し、接続する SSID と PASSWORD の文字列を指定します。UARDECS で使用できる各文字列長は 25 文字までに制限されています。WiFi の規格で認められている本来の文字長より短くなっていますので、ご注意ください。サンプルプログラムでは#define 文で冒頭に”SSID”と”PASSWORD”で記載しています。IP アドレスは、DHCP で自動取得されます。開発に使っている PC が使用している DHCP 機能付きの WiFi ルータの設定にしておくと良いでしょう。専用の WiFi ルータを開発用に設置しておくと、多数のノート開発をするときには効率的です。

シリアルモニター表示しておくと、接続された IP アドレスが表示されますので、ブラウザより、http://接続された IP アドレス/で設定画面を表示させることができます。ここで、実際に使用する WiFi ネットワークや IP アドレスを設定できます。その後、GPIO3 または、D1 端子を開放(プルアップ・オープン)して、ノードをリセットすると、ノーマルモードになります。

5.サンプルプログラムによる使用法の習得

UARDECS のライブラリフォルダに examples フォルダがあり、2 種のサンプルプログラムが用意されています。ArduinoIDE に UARDECS ライブラリを正しく登録した場合は、メニューの [ファイル] ＞ [スケッチ例] ＞ [UARDECS_ESP32C3]から 2 種のサンプルを読み込み、実行することができます。これらのスケッチの注釈を読むことで、UARDECS_ESP32C3 の基本的な使用例を理解することができます。

サンプルプログラムは:

(1) UECS-CCMtransmitter.ino: センサからの計測値(10 秒ごと)と遠隔操作(1 秒ごと)を想定した 2 種の CCM を出力する CCM 送信のサンプルプログラムです。センサとスイッチを装着しない基板だけで動作するデモなので値は全て Web(HTTP)を使って設定し、体験できます。

(2) UECS-CCMreceiver.ino: センサからの計測値(10 秒ごと)と遠隔操作(1 秒ごと)を受信して、設定値による ON-OFF 制御をするサンプルプログラムです。センサとスイッチを装着しない基板だけで動作するデモなので値は全て Web(HTTP)を使って設定し、体験できます。

これらを各基板に書き込んで、WiFi の同じネットワークセグメントに接続し、お互いに CCM の TYPE、ROOM、REGION、ORDER の属性値を一致させておくことで、UECS-CCMtransmitter からセンサの計測データを送信し、UECS-CCMreceiver で受信して設定値以下にならないように出力をオンオフ制御する方法。また、UECS-CCMtransmitter から手動自動の遠隔操作データを送信し、UECS-CCMreceiver で受信して出力に反映する方法などを習得することができます。

実際に、どのような CCM が通信されているかを確かめるには、CCM パケットアナライザソフトウェアである WindowsPC 版 UECS 用パケット送受信ツール(入手先: https://hoshi-lab.info/uecs.org/arduino/uecsrs.html または https://github.com/H-Kurosaki/UecsRS)を使うと便利です。初回起動時に警告が出ることがありますが実行を許可して下さい。

6. UARDECS_ESP32C3 の変数および関数の仕様(参照用)

6.1 定義が必要なグローバル定数と変数(型・変数名・内容)

const char・U_name[ ]・機器の機種名(例えば"Temp controller")。半角英数で 20 文字以内(タグ、ダブルコーテーション使用禁止)。Web ページの初期タイトル。

const char・U_vender[ ]・機器の開発者(例えば"XXX co.")。半角英数で 20 文字以内(タグ、ダブルコーテーション使用禁止)。

const char・U_uecsid[ ]・UECS 研究会が発行する ID 番号(16 進数文字列)。半角英数字 12 文字固定。NODESCAN への応答に使用される。試作機は"000000000000"で可。

const char・U_footnote[ ] ノードの http サーバにアクセスしたときのトップページの下部に表示する文字列。字数制限なし日本語使用可能。タグはそのまま表示されるので不必要に使うとページのレイアウトが崩れるが、逆に利用すればリンクなどを張ることもできる。

char・U_nodename[20]・ノードの http サーバが表示するノードの名称。半角英数字で 19 文字以内、日本語は UTF-8、6 文字以内で使用可能、"&<>"使用禁止。ここで設定された文字列は SafeMode 時の初期値になる。ユーザーが値を変更可能で不揮発性メモリにも保存される。

const int・U_HtmlLine・ノードの http サーバで Node Status 画面に表示する選択肢など設定項目の総数。使用しない場合 0 を指定する。

struct UECSUserHtml・U_html[U_HtmlLine]・Web 画面の表示・設定変数の構造体。{Name,Form format,Unit,Remark,Select,ItemN,value,min,max,digit}として各配列要素を記述する。以下のサンプルのように記載。

『
```
// setting CCM description for web page
const int U_HtmlLine = 2; //Total number of HTML table rows.
// No.1 CCM item
const char NAME1_st[] = "Meas. value";
const char UNIT1_st[] = "";
const char NOTE1_st[] = "センサ計測値";
signed long MeasVal;
// No.2 CCM item
const char NAME2_st[] = "Remote SW";
const char UNIT2_st[] = "";
const char NOTE2_st[] = "遠隔操作";
signed long RemoSW;
const char SW_auto[] = "自動";
const char SW_off[] = "OFF";
const char SW_on[] = "ON";
const char *Menu_rsw[3] = { SW_auto, SW_off, SW_on } ;
const char** dummy = NULL;
// How to show any items in Web page.
struct UECSUserHtml U_html[U_HtmlLine] = {
        // Name, Form format, Unit, Remark, Select,ItemN, value, min, max, digit
        {NAME1_st, UECSINPUTDATA,    UNIT1_st, NOTE1_st, dummy, 0,
        &(MeasVal), -1000, 1000, 1},
        {NAME2_st, UECSSELECTDATA, UNIT2_st, NOTE2_st, Menu_rsw, 3,
        &(RemoSW), 0, 2, 0},
};
```
』

const int U_MAX_CCM・使用する CCM の数を記述する。

struct UECSCCM・U_ccmList[U_MAX_CCM]・使用する CCM の ID を enum 型で定義して、最後に CCMID_dummy を加え、UECSCCM 構造体と共に次のように自動的に設定するのが便利。

『
```
enum {
        CCMID_MeasVal,
```

```
        CCMID_RemoteSW,
        CCMID_cnd,
        CCMID_dummy
        };
const int U_MAX_CCM = CCMID_dummy;
UECSCCMU_ccmList[U_MAX_CCM];
```

』

struct UECSOriginalAttribute・U_orgAttribute・本 UECS ノードの通信状態を記録する変数。スケッチの冒頭部分でグローバル変数として、変数名等を変えずに、
UECSOriginalAttribute U_orgAttribute; のように宣言する必要がある。

6.2 スケッチで実装が必要な関数(関数名・詳細)

void UserInit()・struct UECSCCM の U_ccmList[U_MAX_CCM]で宣言した CCM をスケッチ中で初期化する関数。内部で次の UECSsetCCM()関数で各 CCM を定義する。

void UECSsetCCM()・各 CCM の属性を定義する関数。U_MAX_CCM 数実装する。グローバル定数で値を記述したあと、UserInit()関数中で以下のサンプルのように記載。

『

```
const char ccmName1[] = "Measurement value";
const char ccmType1[] = "MeasVal.mXX";
const char ccmUnit1[] = "";
const char ccmName2[] = "Remote Switch";
const char ccmType2[] = "RemoSW.mXX";
const char ccmUnit2[] = "";
const char ccmNameCnd[] = "NodeCondition";
const char ccmTypeCnd[] = "cnd.mXX";
const char ccmUnitCnd[] = "";
// Description of CCM communication
void UserInit() { // S/!R の項は true が送信 CCM、false が受信 CCM
    // S/!R, Index name, Description, Type name, unit, priority, digit, CCM Lv.
    UECSsetCCM(true, CCMID_MeasVal, ccmName1, ccmType1, ccmUnit1, 29, 1,
    A_1M_1);
    UECSsetCCM(true, CCMID_RemoteSW, ccmName2, ccmType2, ccmUnit2, 29, 0,
    A_1S_0);
```

```
    UECSsetCCM(true, CCMID_cnd, ccmNameCnd, ccmTypeCnd,ccmUnitCnd,29, 0,
    A_1S_0);
}
```

』

CCM レベルについて、本実装では、最短処理間隔(tick)が 1 秒になるため、A_10S_1 および A_1M_1 のレベルで値変化等のデマンドが発生しても、最長 1 秒間の遅延が生じる。さらに、A_1S_0 と A_1S_1 は同義になる。また、B_0、B_1 には対応していない。鮮度管理、CCM 受信が途絶後の有効時間は、各送信間隔(1 秒、10 秒、1 分)の 3 倍に設定されている。

void OnWebFormRecieved()・Web ページの設定画面で http サーバの生成した Node Status 画面で Send ボタンが押され、選択肢などの値が送信された時にこの関数が呼び出される。この関数の実行後、値がフラッシュメモリに記録される。何も実行しないときにも、本体が空で本関数を実装する必要がある。

void UserEverySecond() ・この関数は 1 秒毎に呼び出されるので 1 秒間隔で実行すべき処理を記述できる。この関数実行後に 16520 番ポートに送信条件を満たした通信文が送信される。何も実行しないときにも、本体が空で本関数を実装する必要がある。

void UserEveryMinute()・この関数は 60 秒毎に呼び出されるので 60 秒定間隔で実行すべき処理を記述できる。何も実行しないときにも、本体が空で本関数を実装する必要がある。

void UserEveryLoop()・システムのタイマカウント、http サーバの処理、UDP16520、16529、16521 ポートの通信文をチェックした後、呼び出される関数。呼び出される頻度が高いため、重い処理を記述しないこと。何も実行しないときにも、本体が空で本関数を実装する必要がある。

void UECSloop()・ArduionIDE で通常処理を実施する void loop()の関数中で必ず呼び出さなくてはならない。本ノードで頻繁に処理する内容も記載できるが、void UserEveryLoop()関数と同様に、時間のかかる処理は UECS の定時処理が不安定になるので、void loop()には記載しないこと。できるだけ void UserEverySecond()関数内で記述し、キー押しのチェック程度にとどめること。

void setWiFiCredentials(“SSID”, “PASSOWRD”)・セーフモードの WiFi 接続のための認証情報。ArduionIDE で初期化処理を実施する void setup()中に、かつ、void UECSsetup()の実行前に、必ず記述する必要がある。引数は char*型の文字列ポインタ渡しにする。それぞれ最長 25 文字まで使える。

void UECSsetup()・UARDECS の初期化関数。ArduionIDE で初期化処理を実施する void setup()中に、かつ、void UECSsetup()の実行後に、必ず記述する必要がある。

6.3 スケッチで必要に応じて呼び出せる関数

char* getUARDECSfor()・本 UARDECS が対象とする CPU の文字列を返す。本版は、”UARDECS_ESP32C3”の文字列を返す。

char* getUARDECSon()・本 UARDECS の版(バージョン)文字列を返す。

char* getUECSWiFiSSID()・接続している WiFi の SSID 文字列を返す。

char* getUECSWiFiPASSWORD()・接続している WiFi のパスワード(フレーズ)を返す。情報漏洩防止の観点から製品版にする場合は、得た文字列は内部処理にのみに使用し、可読化を防止すること。

bool UECSWiFiConnected()・WiFi の接続状態を返す。true で接続、false で非接続。

bool UECSSafeMode()・セーフモードの時 true を返す。ノーマルモードでは false になる。

void WiFi.setSleep(bool)・ESP32C3 は電力節約のため WiFi を間欠運転できます。UECS は最短一秒間隔で WiFi での通信を実施する必要があるため、WiFi.setSleep(false);を WiFi 通信の開始前、UARDECS では UECSsetup();の実行前に実行しておくと、通信が安定化します。バッテリー動作など間欠通信を可能にして UECS を使用する場合には、引数を true で呼び出し、適宜スケッチを記述します。

7.おわりに

UARDECS_ESP32C3 を使用するために最低限必要な情報を記載しています。本説明書に記載のない詳細事項については、UEARDECS_MEGA および UARDECS_PICO の説明書(https://github.com/H-Kurosaki/UARDECS_Documents)と、本ミドルウェアに添付のサンプルプログラムを参照願います。