UECS ノード開発用ミドルウェア
UARDECS　Ver1.2.0
UARDECS_MEGA　Ver1.2.0
説明書
1.0 版

Writen by　Ken-ichiro Yasuba

since 2013. 4. 17.

Last Updated by Hideto Kurosaki

2018. 10. 9.

1　はじめに

UARDECS(ユアルデックス)は近年普及が進むオープンソースハードウェアである Arduino を利用してユビキタス環境制御システム　(UECS)のノードを作成するために開発されたミドルウェアであり、必要最低限の機能を搭載したノードの開発を容易なものとし，施設園芸の環境制御システムの構築を今まで以上に簡単にすることを目的としています．このミドルウェアは、UECS の通信実用規約"1.00-E10"　に対応した通信文の送受信管理とマイコンボードに搭載した http サーバを利用した設定画面の生成、設定値の不揮発性メモリへの自動読み書き等を行うことができます。

UARDECS で作成した気象観測ノード

制限事項

○UARDECS はメモリ搭載量の少ないマイコンボードを利用するため機能に制限があります
○本ソフトウェアの利用は、同封のライセンス条項に同意した上で行ってください。

２　対応機種

UARDECS はターゲットとするマイコンボードを以下の機種と Shield の組み合わせとしています。さらに、最低限の開発環境を構築するには、これらのマイコンボードに加えて、開発用 PC、USB ケーブル、LAN ケーブル、電源用 AC アダプタ(DC ジャックに給電する場合 GF12-US0913、USB 端子から給電する場合 AD-L50P100 など)を準備する必要があります。

◎UARDECS の動作機種（これ以外にも互換機を使用可能※1）

Arduino UNO+Ethernet Shield R3 (W5100)
Arduino MEGA+Ethernet Shield R3 (W5100)
Arduino Ethernet R3 (W5100)
Arduino UNO+Ethernet Shield 2 (W5500)
Arduino MEGA+Ethernet Shield 2 (W5500)
Arduino UNO+WIZ550io(W5500)
Arduino MEGA+WIZ550io(W5500)

(Arduino Leonardo Ethernet はコンパイル可能ですがメモリ不足により非推奨)

◎UARDECS_MEGA の動作機種（これ以外にも互換機を使用可能）

Arduino MEGA+Ethernet Shield R3(W5100)
Arduino MEGA+Ethernet Shield 2(W5500)
Arduino MEGA+WIZ550io(W5500)

(Arduino UNO ではメモリ不足により非推奨)

現在 UARDECS では W5100 と W5500 というイーサネットコントローラーIC に対応しています。Arduino 純正機では W5100 搭載機が Ethernet Shield と Arduino Ethernet にあたりますが、両機種とも 2018 年現在では販売が終了しており、入手可能なのはサードパーティ製の互換機になります。互換機でも純正品とソフトウェア上では全く同じように動作しますが、MAC アドレスが付属しない商品では、商業利用する場合は別途 MAC アドレスを入手する必要があります。W5500 は W5100 より高速(UNO 搭載時でおよそ 1.7 倍)に動作し、信頼性が向上しています。この搭載機種には Ethernet Shield 2 と Arduino Leonardo Ethernet が該当します。しかし、2018 年現在では販売終了という情報があり、対策を検討中です。W5500 を搭載し、類似の機能を持つモジュールに WIZ550io が存在します(１０章に関連記述あり)

※1「Ethernet Shield for Arduino V2.2」として販売されている DFROBOT 社の黒色のシールドは W5100 搭載機なので注意してください。

3　UARDECS と UARDECS_MEGA の違い

このライブラリには UARDECS と UARDECS_MEGA の 2 種類が収録されています。それぞれインクルード時のファイル名を“Uardecs_mega.h”,”Uardecs_mega.h”とすることで利用可能です。ソースコードでは UARDECS_MEGA は UARDECS の上位互換であり、同じソースコードをコンパイルできます(UARDECS_MEGA→UARDECS への移行は不具合が生じることがあります)。UARDECS_MEGA では Web 上からユーザーが設定できる項目が大きく増えています。

	UARDECS	UARDECS_MEGA
主な対応機種	Arduino UNO ※1 Arduino MEGA	Arduino MEGA ※2
消費メモリ	少	多
CCM の Type 文字列の書き換え	プログラムにハードコーディング(ソースコードに直接記述)	Web 上で編集可能※3
CCM の room-region-order の扱い	Web 上でノード単位に設定	Web 上で CCM ごとに別々に設定可能
設定可能な CCM 送信レベル	A_1S_0 A_10S_0 A_1M_0 S_1S_0	A_1S_0 A_10S_0 A_10S_1 ※4 A_1M_0 A_1M_1 ※4 S_1S_0
入力可能な URL 文字列長	300byte	500byte ※2
EEPROM の利用範囲	0x320 以降	0x9AC 以降 ※2
その他		UARDECS 用のソースコードをそのままコンパイル可能

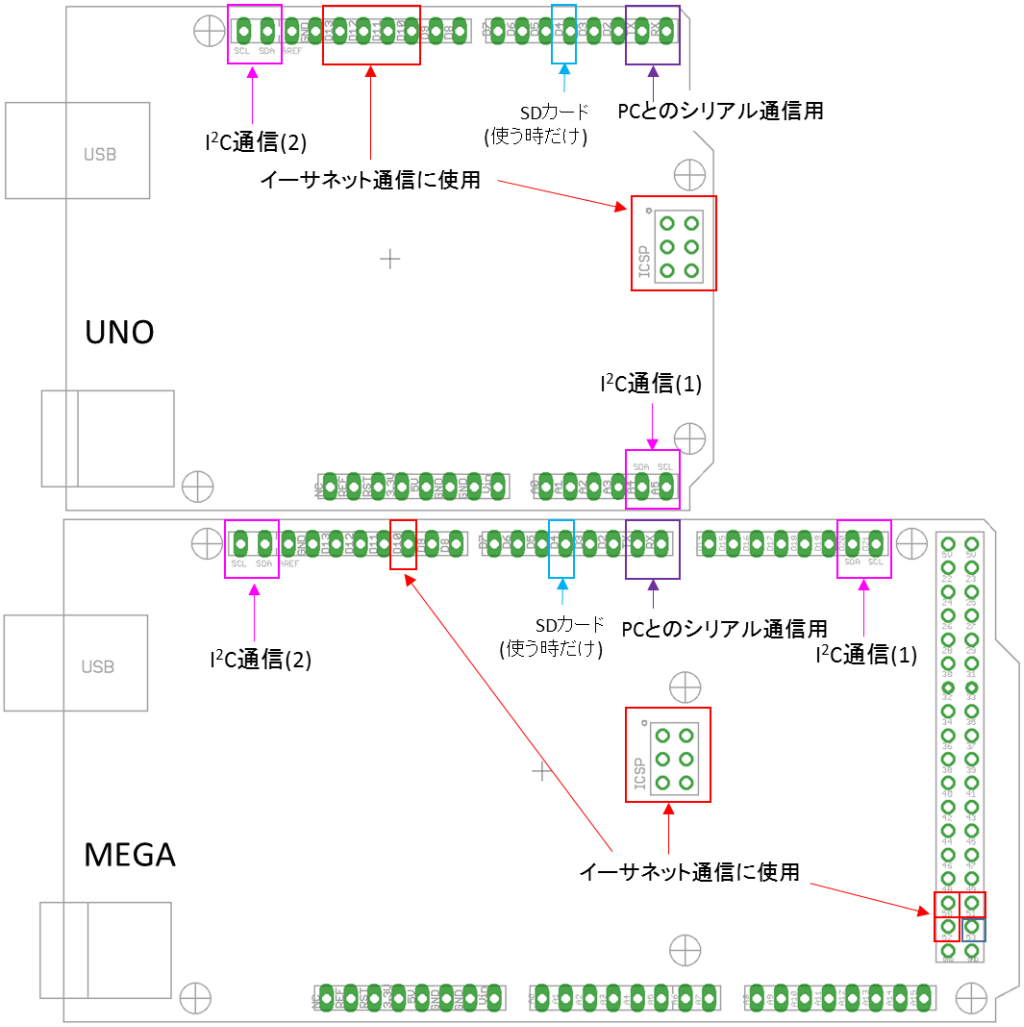
※1 UNO ではユーザーが利用できるのはプログラム用フラッシュメモリの 15%に限定される。フラッシュメモリの消費量だけでなく、SRAM の残量にも注意すること。

※2 UNO でコンパイルした場合、メモリ削減のため動作が異なるが非推奨

※3 プログラム上に記述した Type 文字列はデフォルト値という解釈になる

※4 値が変化した場合は 1 秒以内に送信される。最大送信頻度は 1 秒間隔を超えることはないので A_1S_1 は実装できない。

4　Arduino の注意すべき仕様



１）Arduino で UECS 用のノードを作る場合、入出力ピンに様々なデバイスを接続することになりますが Arduino には仕様上特殊な役割を持つピンがあり、注意が必要なのでその一部を紹介します。まず、D0,D1 ピンは主に PC とのシリアル通信に使用されており、他の用途に使うとスケッチの書き換えやデバッグ時のシリアル出力ができなくなるので注意が必要です。また、イーサネットシールドは D10 と ICSP と書かれた 6 ピンの端子を使用しますが、ICSP の一部は UNO で D11-D13、MEGA では D50-D52 と共有されていますのでこれらのピンを特に意図が無い限り使わないことをお勧めします。D4 ピンはイーサネットシールド上の SD カードスロットを使用する場合のみ専有され、他の用途に使えなくなります。また、I²C 通信はセンサなどで多用されますが、UNO と MEGA では使用されるピンの位置(UNO:A4,A5 MEGA:D20,D21)が異なります。このピンの違いは USB ポートの側にある I²C 通信(2)のピンを使うことで解決できます(内部で I²C 通信(1)と結線されています)。IDE Ver1.7.8 時点で MEGA の D53 は使用されていませんがイーサネット初期化時に High が出力されることを確認しました(IDE のバグ?)。

２）アナログ入力ピンをデジタル入出力として使う

Arduino のアナログ入力ピンアナログ入力用として活用しない場合、例えば以下のように指定すると普通のデジタル I/O として使えます(この機能は Atmega 系の CPU 搭載機種のみ利用可能)。

記述例

```
//出力用として利用
pinMode(A0, OUTPUT);
digitalWrite(A0, HIGH);

//入力用として利用
pinMode(A0, INPUT_PULLUP);//INPUT_PULLUP も設定できます
i=digitalRead(A0);
```

Arduino ではピンに通し番号が付いていますが、アナログ入力ピンの番号は各機種のデジタル入出力ピンの後に設定されています。すなわち UNO では 14 以降、MEGA では 54 以降がアナログ入力ピンとなります。

３）ウォッチドッグタイマ(WDT)実装の時の注意点

Arduino の CPU にはウォッチドッグタイマ(WDT)機能が内蔵されており、IDE 付属のライブラリで利用できますが、MEGA のブートローダーにはバグがあり WDT を使用すると完全なデッドロックになり、リセットボタンでも復帰できない状況に陥るので MEGA で内蔵 WDT の使用はできません。必要であれば専用 IC を外付けして下さい。（１１章に関連記事あり）

４）PROGMEM 修飾子について

Arduino ではプログラム用のコードはフラッシュメモリに、変数は SRAM に格納されますが、SRAM の容量が少ないため効率良く利用しなければなりません。特に文字列を扱う場合、普通に宣言すると SRAM に格納されてしまい容量を圧迫しますので、変更の必要がない文字列をフラッシュメモリに格納するために UARDECS は PROGMEM を多用しています。
PROGMEM 指定された文字列を操作するには専用の関数を使用する必要があります。

５）I^2C 通信の自動プルアップ

I^2C 通信を行う場合、信号線のプルアップが必要ですが、Arduino IDE の標準ライブラリは使用するピンを内蔵回路でプルアップするのでプルアップ抵抗は不要です。ただし、強制的に 5V でプルアップされるので、3.3V 専用のデバイスは電圧レベルの変換が必要です。

６）W5100 の不具合とパッチについて

W5100 のドライバには最新版(本稿執筆時点で Ver2.0.0)でも不具合があり、数時間もしくは数日稼働させるとフリーズすることがあります。この原因は Ethernet ドライバの中に含まれる socket.cpp の UDP 送信部分にあります。参考としてライブラリ Ver2.0.0 のソースコードを示しますが、問題の箇所のタイムアウト判定にバグがあり、タイムアウトした場合に想定とは異なる戻り値が W5100 から返されるために無限ループに陥ります。この問題を修正するパッチを同封していますので必ず使用してください。(ライブラリのバージョンが違うとパッチが機能しません。また、W5500 にはこの問題はありません)

```
bool EthernetClass::socketSendUDP(uint8_t s)
{
	SPI.beginTransaction(SPI_ETHERNET_SETTINGS);
	W5100.execCmdSn(s, Sock_SEND);

	/* +2008.01 bj */
	while ( (W5100.readSnIR(s) & SnIR::SEND_OK) != SnIR::SEND_OK ) {
		if (W5100.readSnIR(s) & SnIR::TIMEOUT) {//←問題の箇所
			/* +2008.01 [bj]: clear interrupt */
			W5100.writeSnIR(s, (SnIR::SEND_OK|SnIR::TIMEOUT));
			SPI.endTransaction();
			//Serial.printf("sendUDP timeout¥n");
			return false;
		}
		SPI.endTransaction();
		yield();
		SPI.beginTransaction(SPI_ETHERNET_SETTINGS);
	}

	/* +2008.01 bj */
	W5100.writeSnIR(s, SnIR::SEND_OK);
	SPI.endTransaction();

	//Serial.printf("sendUDP ok¥n");
	/* Sent ok */
	return true;
}
```

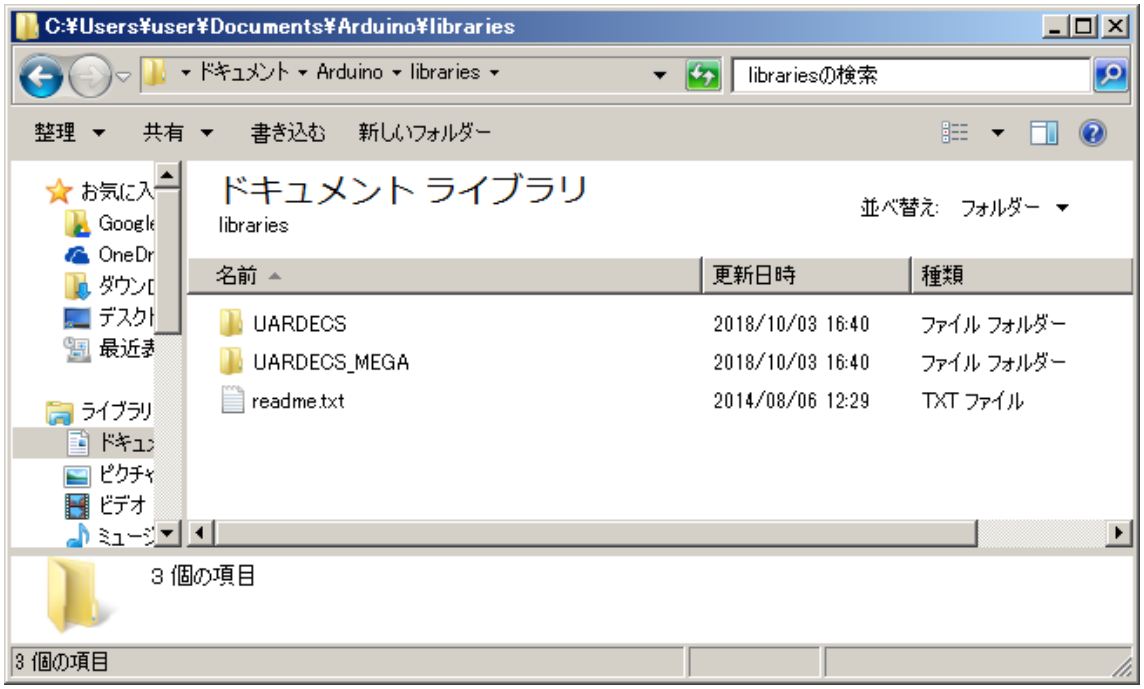
5　開発環境の構築

Windows PC を利用して開発環境を構築する例を説明します。UARDECS Ver1.2.0 を使用するには開発用 PC に Arduino IDE Ver1.8.7 以降をインストールする必要があります。（ダウンロード先：https://www.arduino.cc/）

1）Arduino の開発環境をダウンロードします．インストーラ付きの Windows 用、バージョンは Ver1.8.7 以降とします。これを開発用 PC にインストールします。

2）UECS ノード開発用ミドルウェア UARDECS の zip ファイルを解凍します．

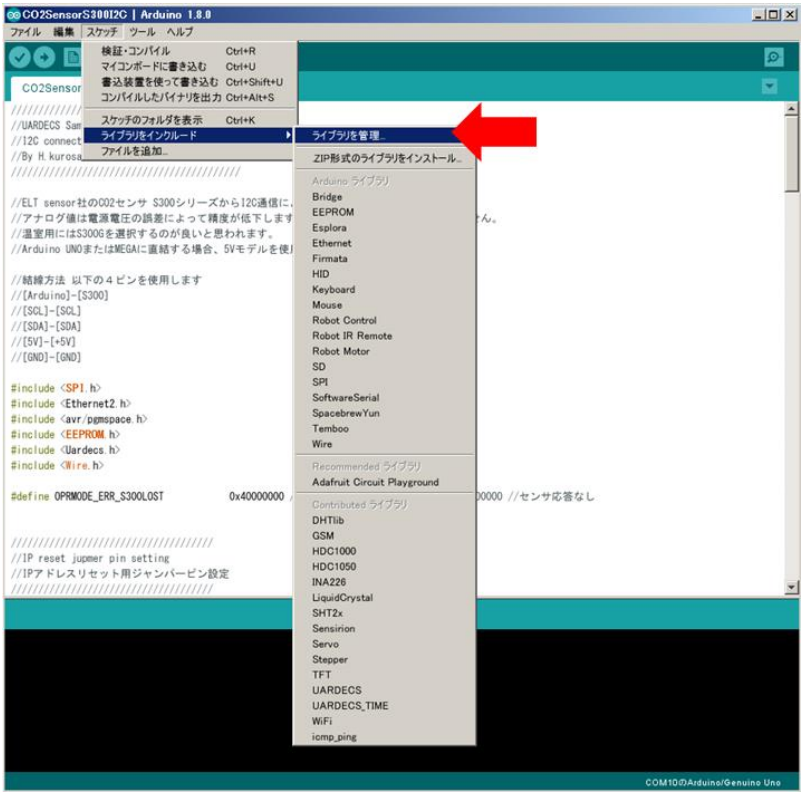
3a）W5500 搭載機(Ethernet Shield2 など)を利用して開発を行う場合、マイドキュメント/Arduino/libraries というフォルダがあるので zip ファイルに入っていた W5500 の中の UARDECS と UARDECS_MEGA をフォルダごとコピーします。旧バージョンがインストールされている場合、旧バージョンは削除してから新しいファイルをコピーしてください。



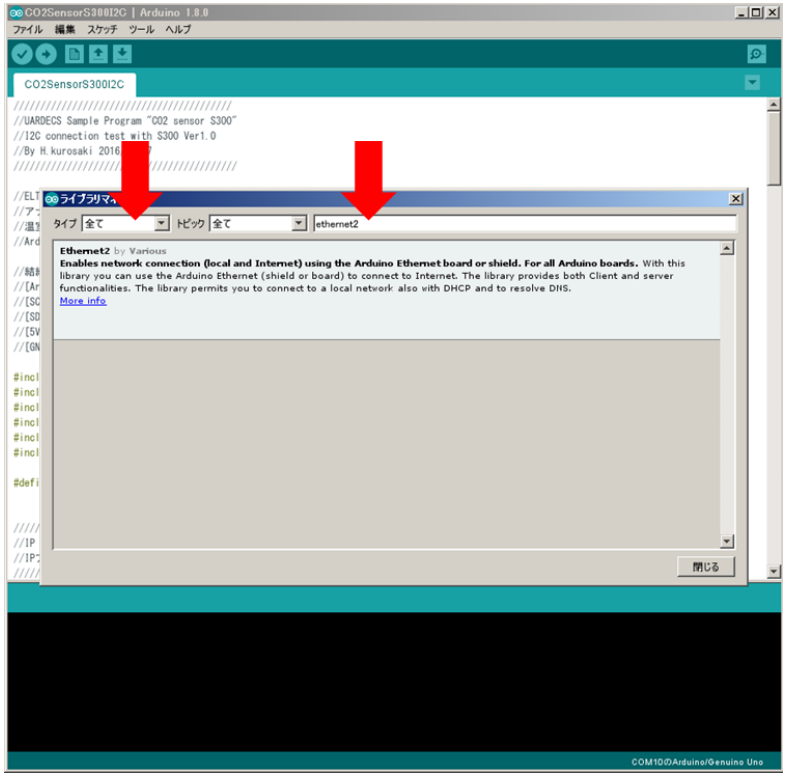
マイドキュメント/Arduino/libraries の中に UARDECS と UARDECS_MEGA をコピーした例

次に、以下の手順で Ethernet2 ライブラリをインストールします。

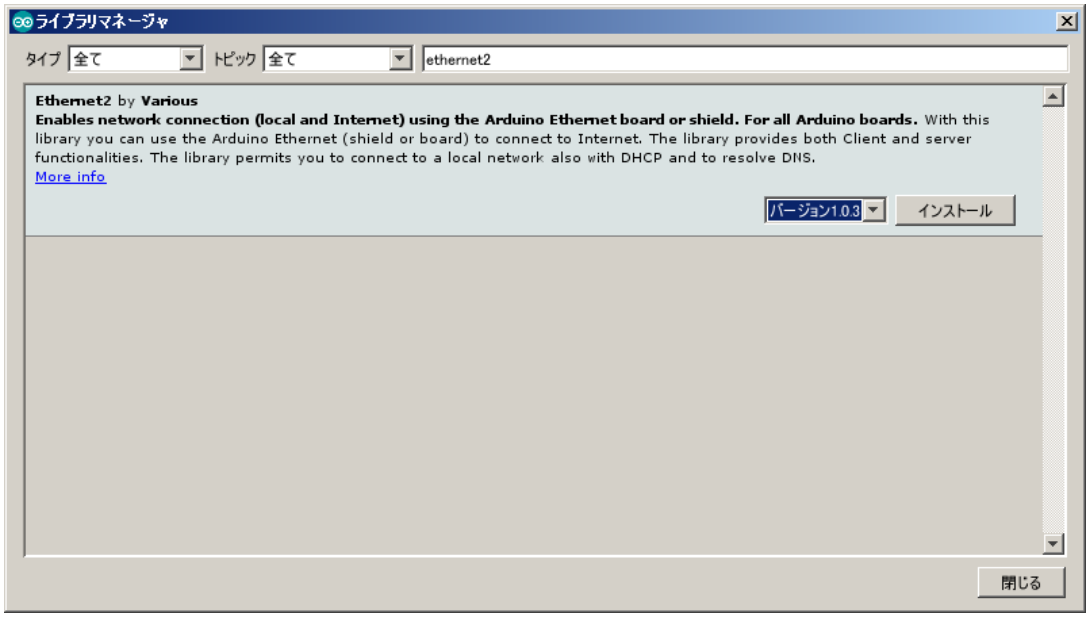
(1)IDE を起動し「スケッチ」→「ライブラリをインクルード」→「ライブラリを管理」を選ぶ



(2)下図のようになるので、タイプに「全て」検索欄に「ethernet2」と入力する



(3)Ethernet2 by Varius・・・というライブラリが出てくるのでそれを選ぶ。水色の部分をクリックするとインストールボタンが出るので押すバージョンは通常は最新バージョンを選ぶ(同封の Wiz550io 用任意パッチを使う場合 1.0.3 を選ぶ)

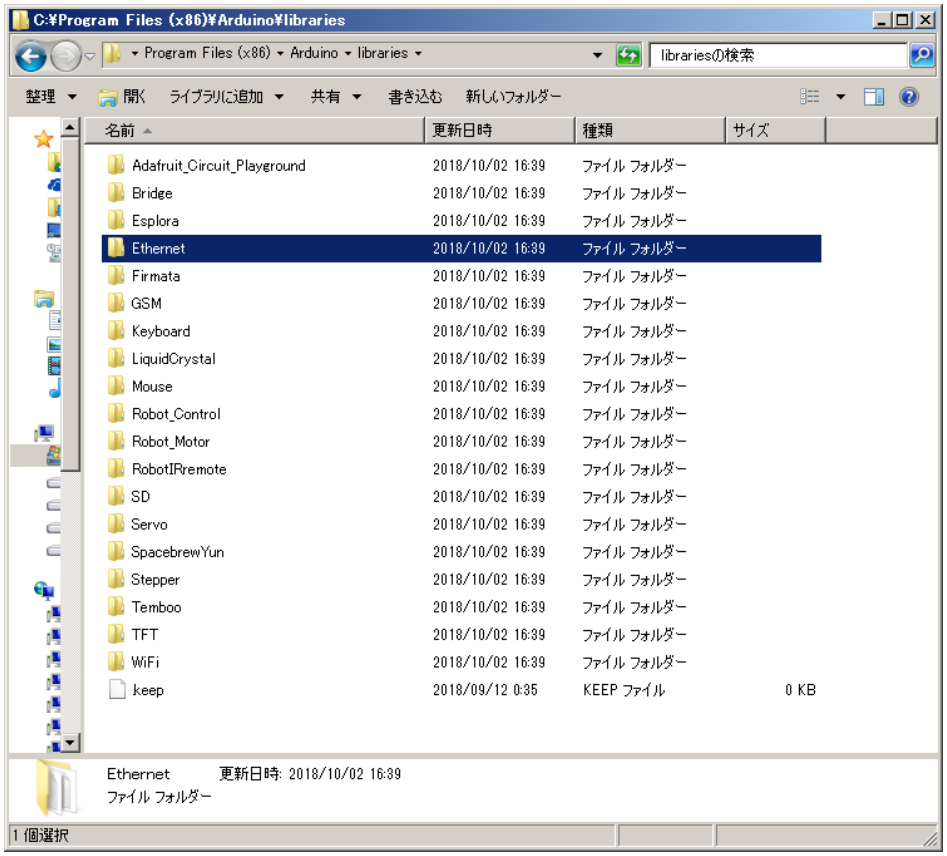


(4)IDE1.8.7 ではライブラリのインストールフォルダがドキュメントフォルダ以下の Arduino/libraries の中になっているので、ここに Ethernet2 フォルダが出現していれば成功です。

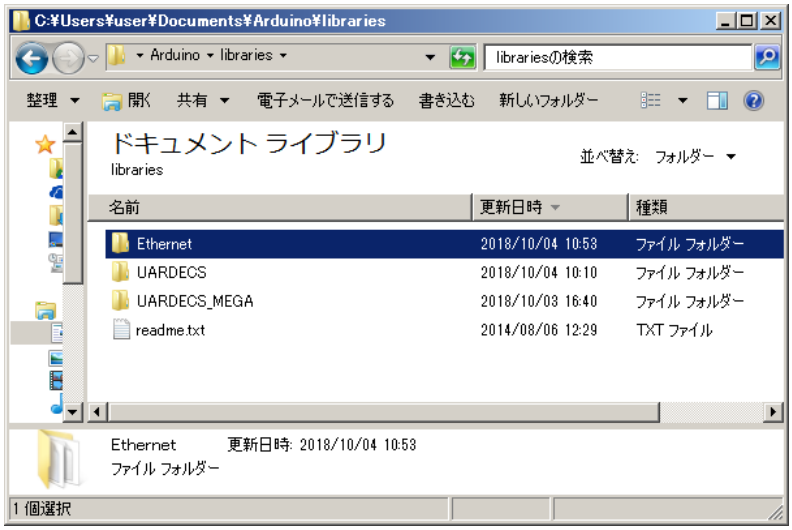
3b) W5100 搭載機(Ethernet Shield, Arduino Ethernet など)で開発を行う場合、マイドキュメント/Arduino/libraries というフォルダがあるので zip ファイルに入っていた W5100(旧機種用)/libraries の中にある UARDECS と UARDECS_MEGA をフォルダごとコピーします。旧バージョンがインストールされている場合、旧バージョンは削除してから新しいファイルをコピーしてください。

※W5500 用と W5100 用のライブラリでは収録されているサンプルプログラムに違いがあります

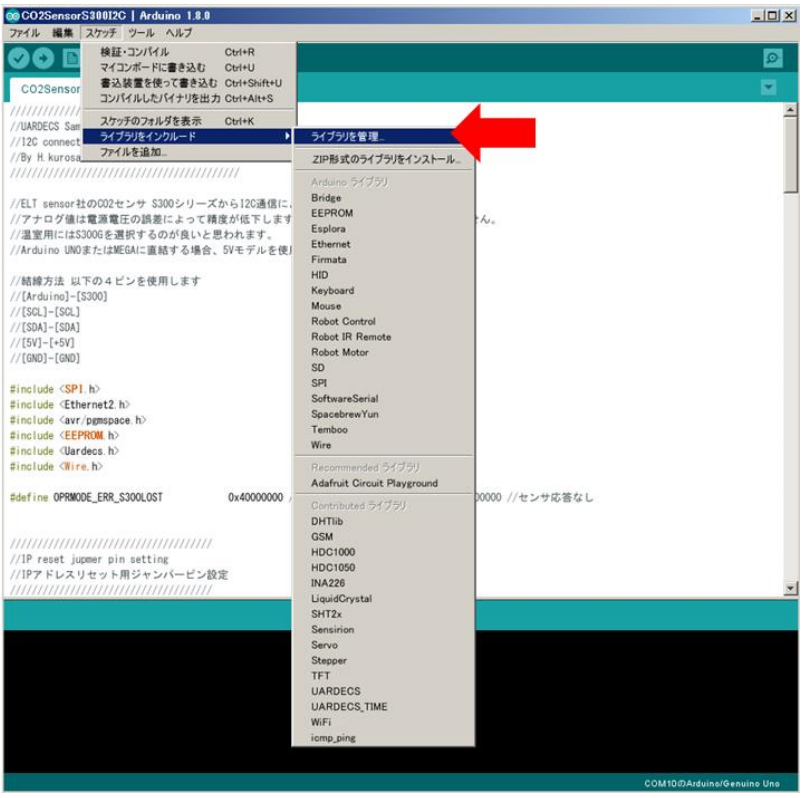
次に、以下の手順で Ethernet ライブラリの対応バージョンとパッチをインストールします。



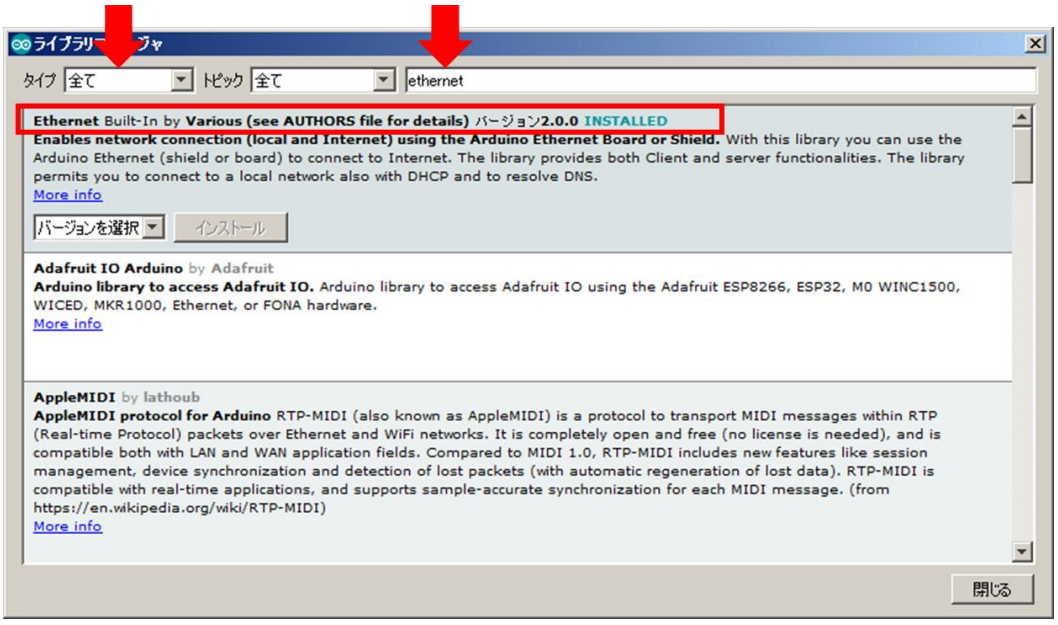
(1)最初からインストールされている Ethernet ライブラリを探します。インストーラ付きの IDE を使っている場合は、”C:¥Program Files (x86)¥Arduino¥libraries”の中にあります。ここに Ethernet というフォルダがあるのでフォルダごと、ここからコピーします。



(2) Ethernet ライブラリをマイドキュメント/Arduino/libraries の UARDECS と同じフォルダの中にペーストします(同じライブラリがある場合、こちらのフォルダにある方が優先して使われます)。

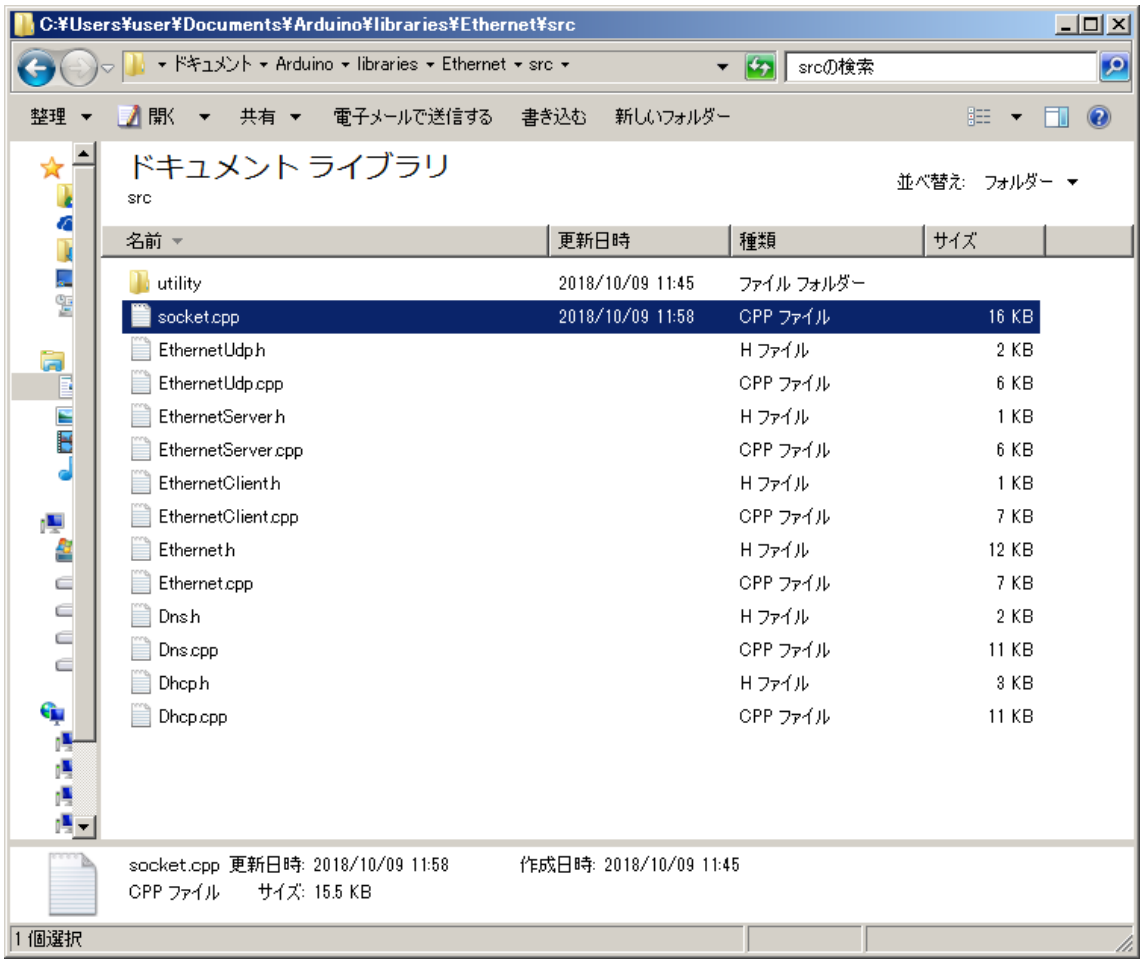


（3）IDE を起動し「スケッチ」→「ライブラリをインクルード」→「ライブラリを管理」を選びます。



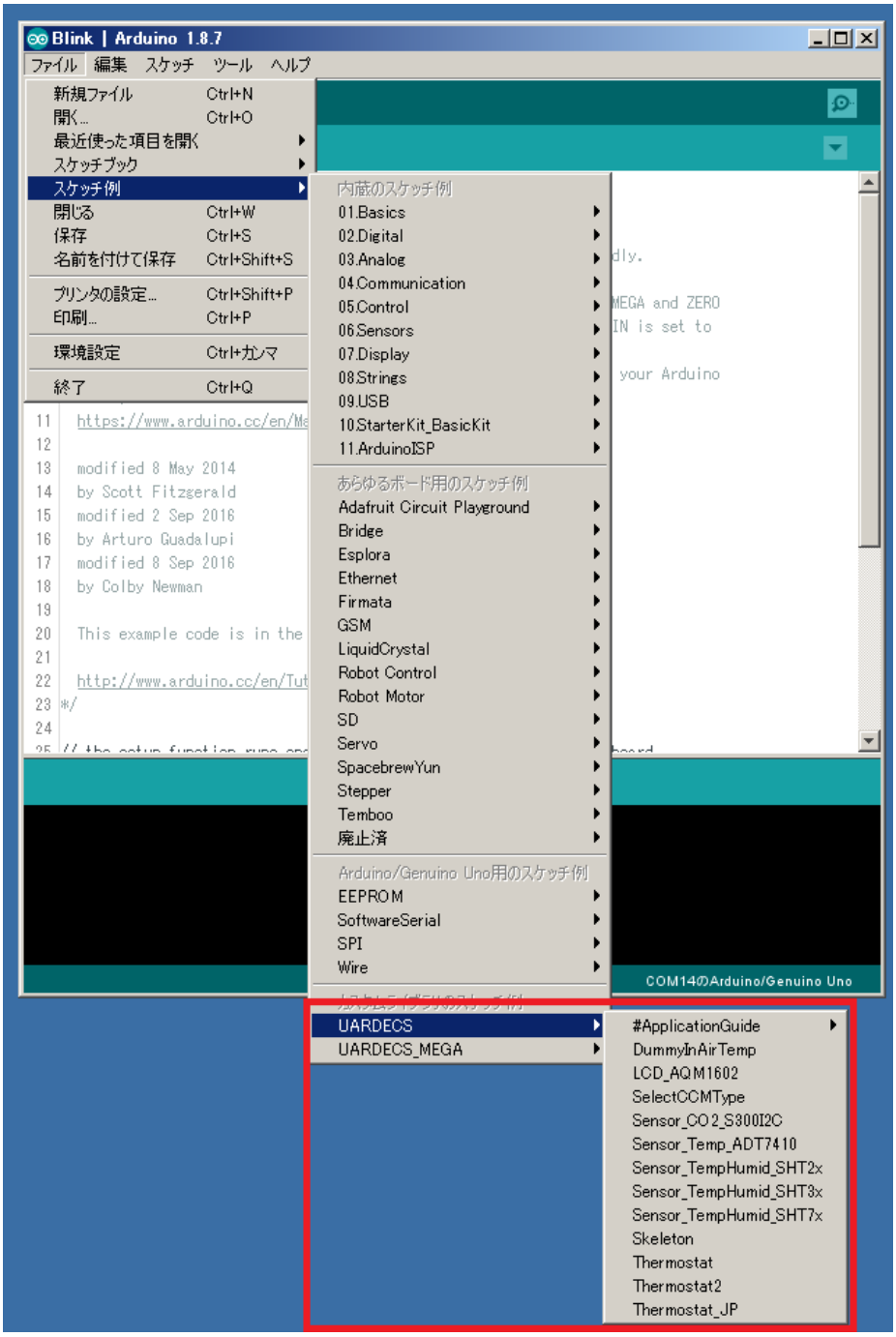
（4）ライブラリマネージャという画面が出るので、タイプに「全て」、検索欄に ethernet と入力すると Ethernet Built-IN by Various・・・というライブラリが出てくるのでバージョンが 2.0.0 であることを確認します。

(5)IDE1.8.7 ではバージョン 2.0.0 が最初からインストールされています。もし、他のバージョンが検出された場合、バージョンを選択→バージョン 2.0.0 を選んでインストールして下さい。

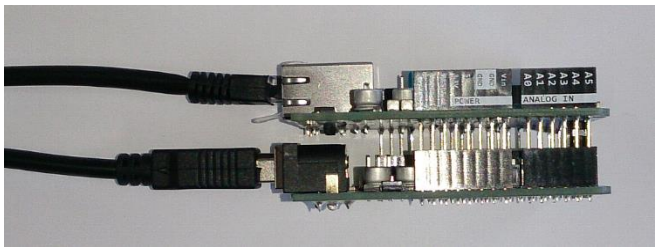


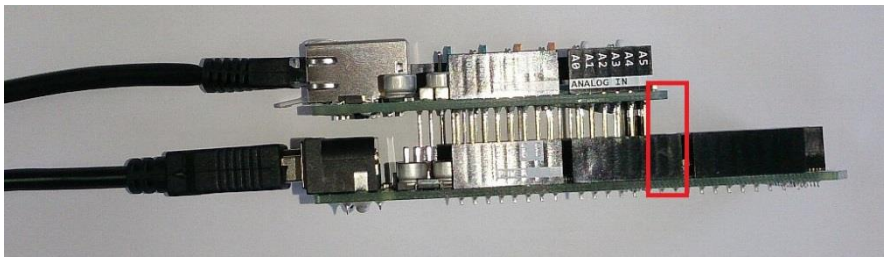
(6)次に、再びマイドキュメント/Arduino/libraries フォルダに入ります。先程コピーした Ethernet フォルダを開き、その下の src フォルダに入ると”socket.cpp”があります。UARDECS の″W5100 用 Ethernet パッチ(不具合対応)″フォルダにある、Ver2.0.0 用の″socket.cpp″で上書きします。（ただし、バージョンアップの知らせが出た時、その指示通りにこのライブラリをバージョンアップするとパッチが上書きされて効果が失われるので注意すること）

４）正常に UARDECS がインストールされている場合、Arduino IDE を起動したとき図の位置に UARDECS のサンプルプログラムのリストが表示されます。



6 Arduino の準備と開発用 PC の IP アドレス設定





１）Arduino に Ethernet Shield を装着します。MEGA では赤枠の部分で 2 本ピンが余りますが正常です。

２）Arduino に USB 端子と LAN ケーブル(普通のストレートケーブルで良い)を接続し、両方とも開発用 PC に接続します。正常に認識されればシリアル通信用のドライバがインストールされます。（この状態では Arduino は PC の USB ポートから給電されて動作します）

３）開発用 PC の Arduino に接続した LAN ポートの IP アドレスを以下の値に設定し直します。

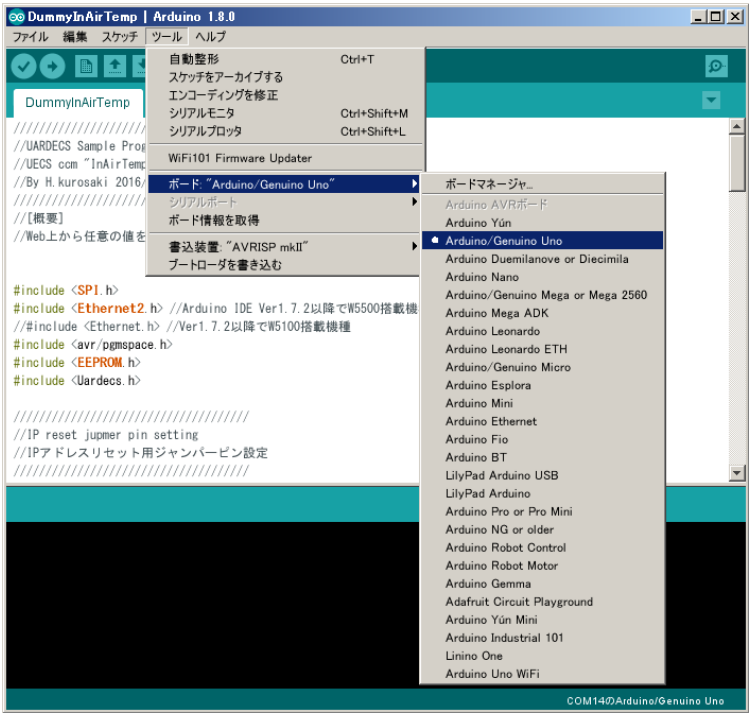
IP アドレス :192.168.1.1
サブネットマスク:255.255.255.0
デフォルトゲートウェイ:変更不要
DNS サーバーアドレス:変更不要

方法がわからない方は以下のリンクを参考にしてください。

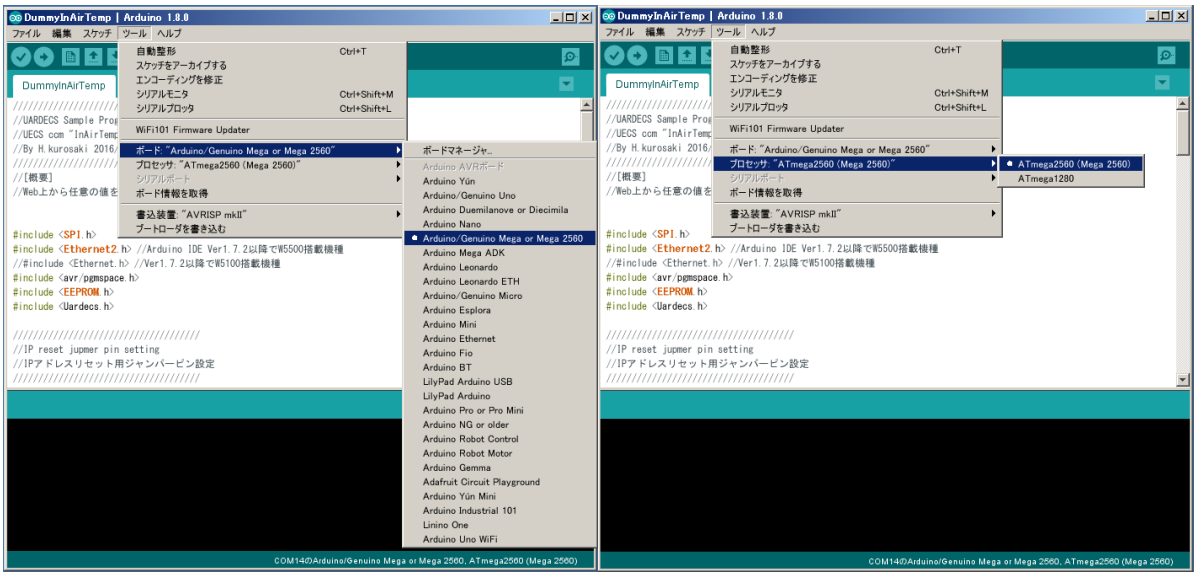
パソコンの IP アドレスを手動で設定する方法

注意：設定を変更するとインターネットに接続できなくなることがあります。不安な方は後で設定を戻せるように変更前の状態をメモして下さい。

注意：複数のイーサネットポート(LAN ポートが２つある、LAN ポートと無線 LAN 機能があるなど)がある PC では UDP パケットのブロードキャスト時に混乱を招くことがあります。通信に問題が生じる場合、不要な LAN ポートのケーブルを抜くなどして無効にして下さい。

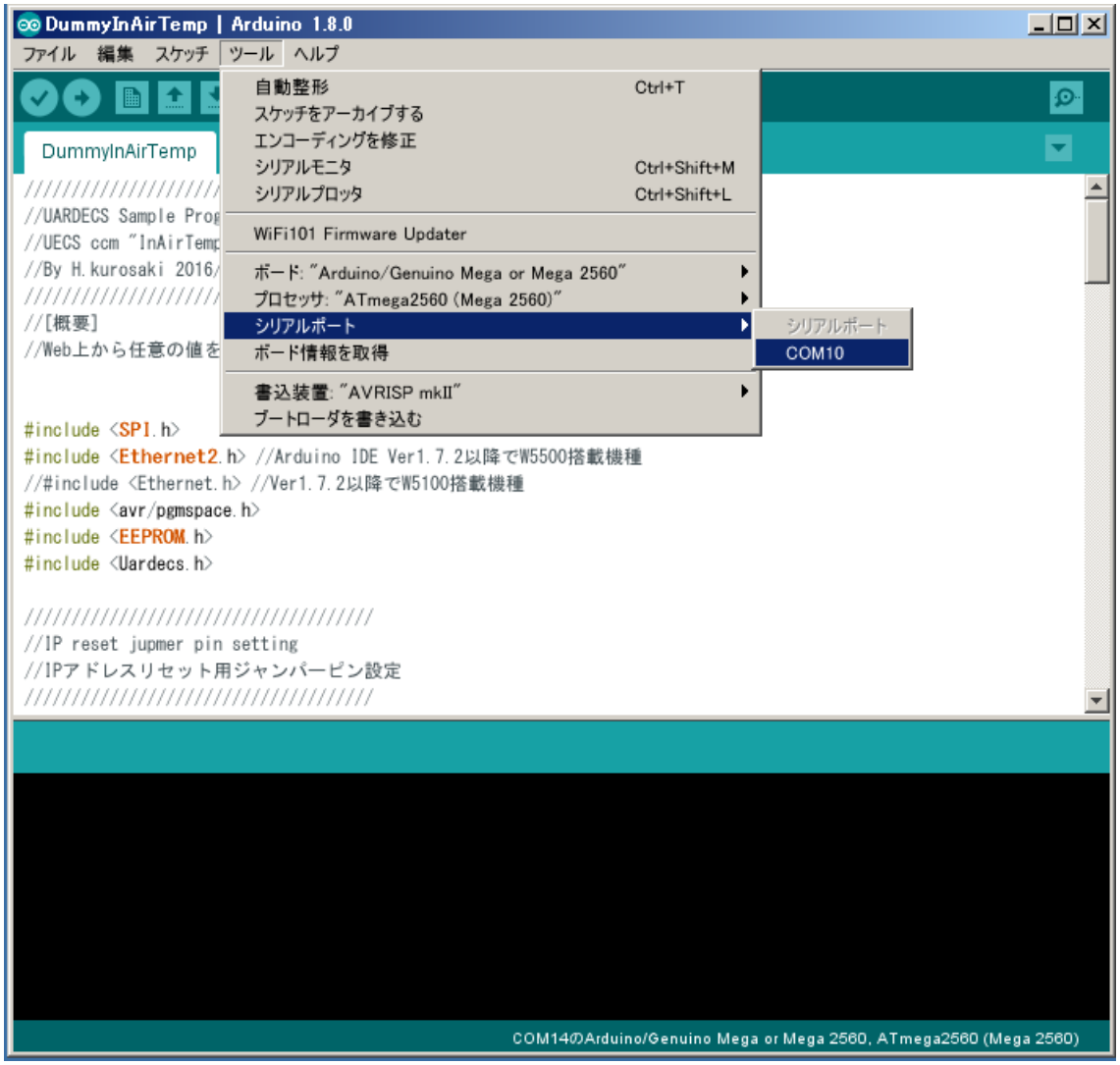


ボードの指定（UNO の場合 Arduino/Genuino UNO を選択）



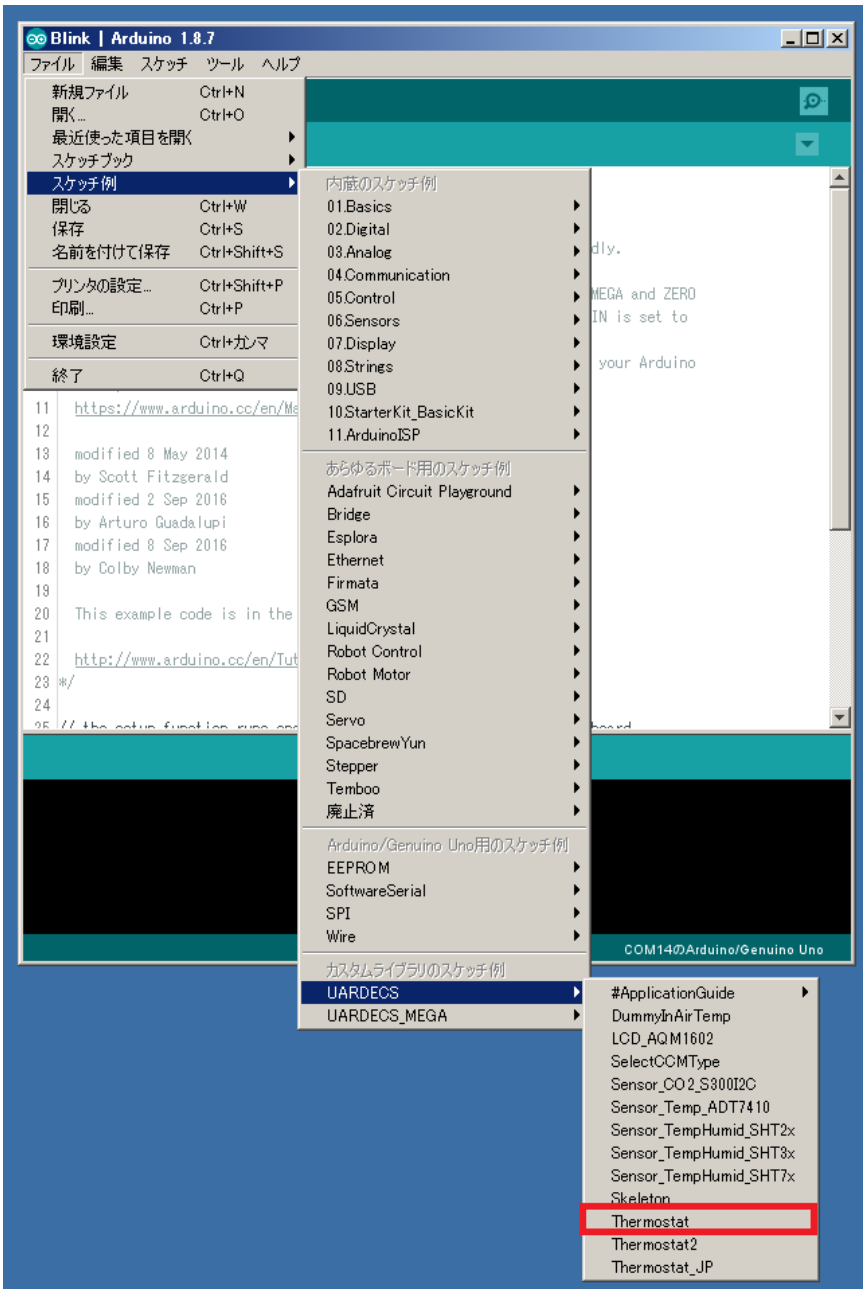
ボードの指定（MEGA の場合 Arduino/Genuino MEGA or MEGA 2560 を選択）

4）Arduino IDE を起動し、ボードの種類を指定します。MEGA を使用する場合、プロセッサも指定する必要があります(現在販売されている MEGA は全て 2560 です)。

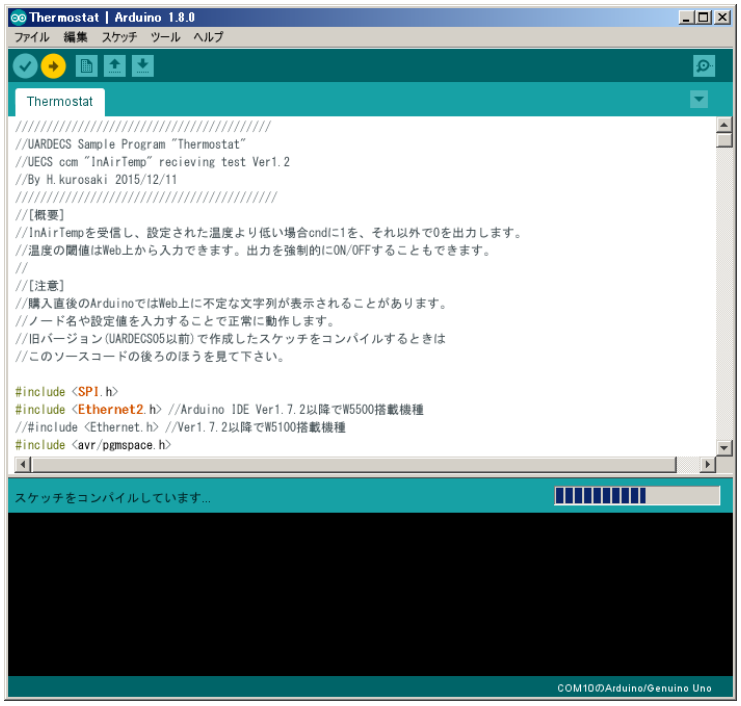


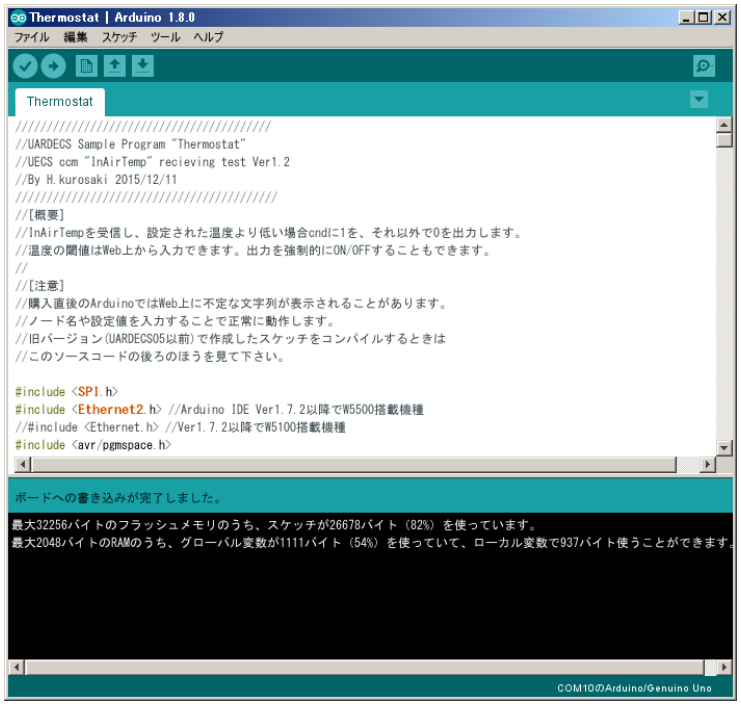
５）PC に Arduino が接続されているシリアルポートを指定します。ポート番号の表示は PC によって異なります。一般に、下の方に表示されるポートが有効になることが多いようです。間違ったポートを指定すると、マイコンボードにスケッチを書き込むことができません。同一機種でも違う Arduino を接続すると、この番号は変動するので、再設定が必要になります。

7　サンプルプログラムの実行(UARDECS)



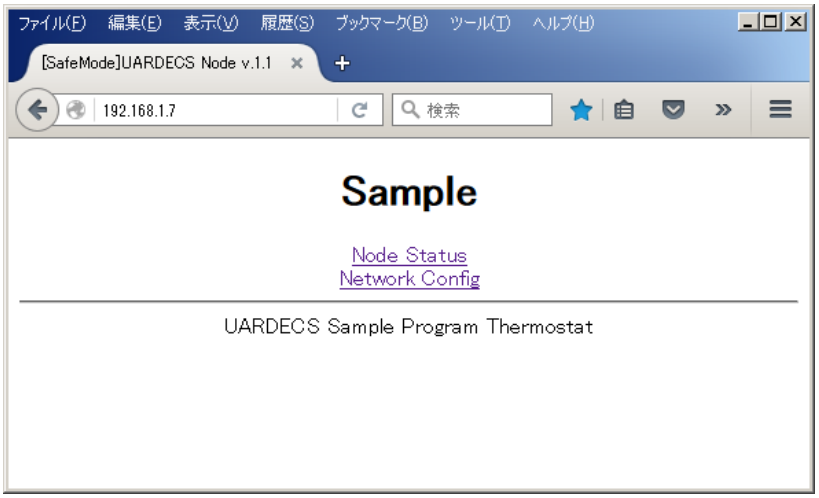
1）スケッチの例→UARDECS の中から試しに Thermostat を呼び出してみます





２）IDE 左上の→ボタンをクリックするとコンパイルが始まり、プログラムが自動的に Arduino に書き込まれます。正常に終了すると「マイコンボードへの書き込みが完了しました」と表示されます。

※コンパイルを連打するとタイミングによってはエラーが出ることがあります



３）プログラムを書き込んだノードの動作を確認します。Web ブラウザを用いて”192.168.1.7”にアクセスすると、ノードが正常に動作している場合、上の画面が表示されます。工場出荷時の Arduino を使用した場合、自動的に SafeMode に入ります。

SafeMode について：

工場出荷時の IP アドレスが未設定な状態か、設定した IP アドレスが分からなくなった時に、SafeMode ジャンパーを設定して起動すると SafeMode に入ります。この状態では Web 画面のタイトルの部分に[SafeMode]と表示されます。

SafeMode は時にはノードの設定にかかわらず、以下の IP アドレスが設定されます

IP アドレス　　：192.168.1.7

サブネットマスク：255.255.255.0

SafeMode ジャンパーについて：

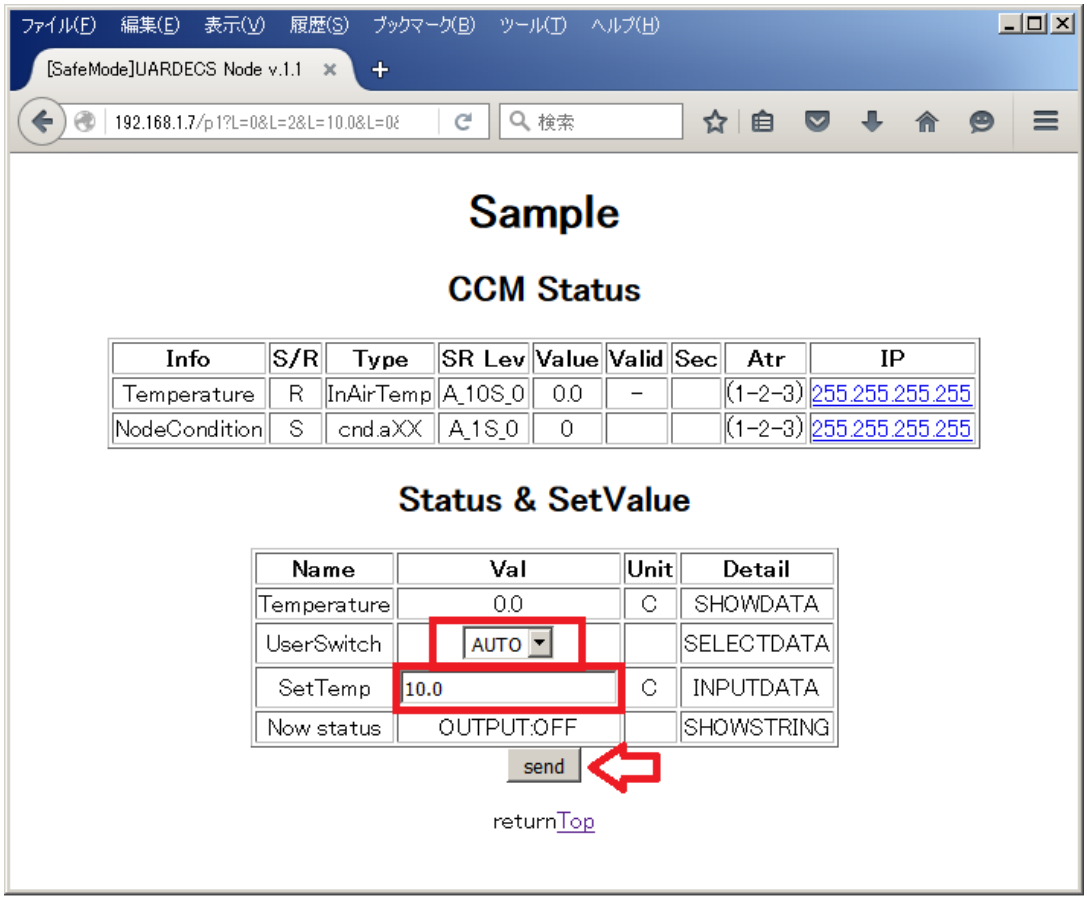
Thermostat のサンプルスケッチ中に

```
const byte U_InitPin = 3;
```

と書かれているのが SafeMode ジャンパー用に使用する Arduino のピンです。最初に D3 が指定されていますが、別のピンにも変更可能です。

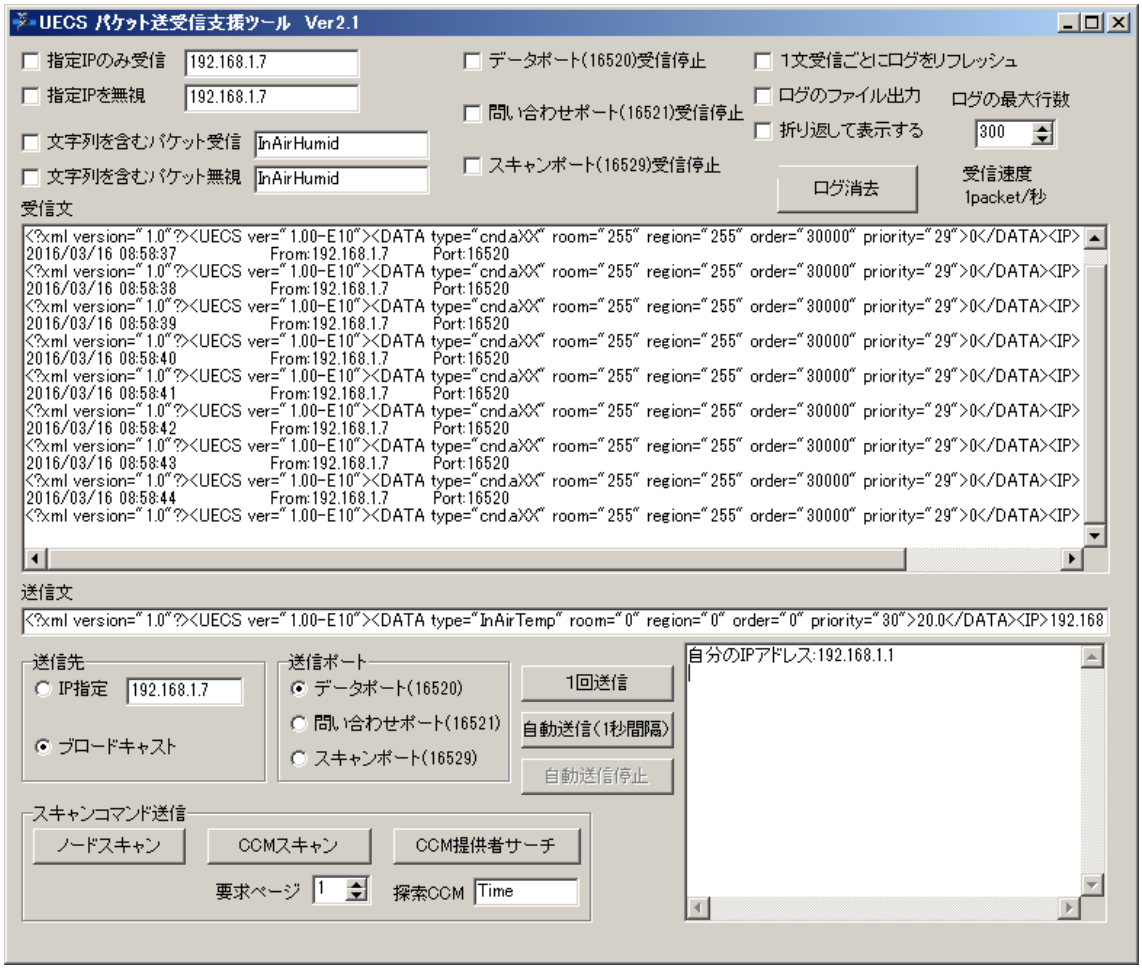
```
const byte U_InitPin_Sense=HIGH;
```

と書かれているのが検出条件で、SafeMode ジャンパーがこの値を示した時に SafeMode に入ります。SafeMode ジャンパーは自動プルアップされる(すなわち HIGH になる)ので、Thermostat のサンプルスケッチは D3 に何も接続しない場合、強制的に SafeMode で動作することを示しています。SafeMode を抜けたい場合、Arduino の D3 と GND 間を 1kΩの抵抗を介してつなぐか、U_InitPin_Sense=LOW;に書き換えて下さい。

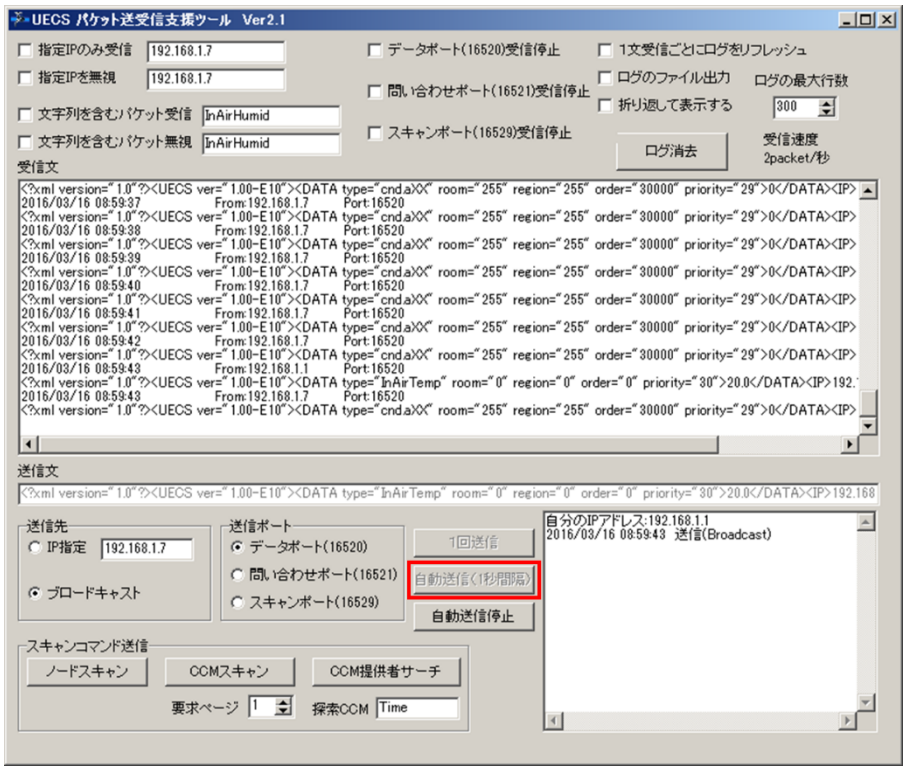


４）Web ブラウザの画面から Node　Status に入るとサーモスタットの動作を設定できます。試しに UserSwitch に AUTO を、SetTemp に 10 を指定して send ボタンを押すと、設定が書き込まれます。ここで設定した値は不揮発性メモリに書き込まれ、ノードの電源を切っても設定値が残ります。ただし、SafeMode　では電源投入後に値の復帰は行われず、初期値が入力されます(SafeMode 中でも値の保存は可能)。

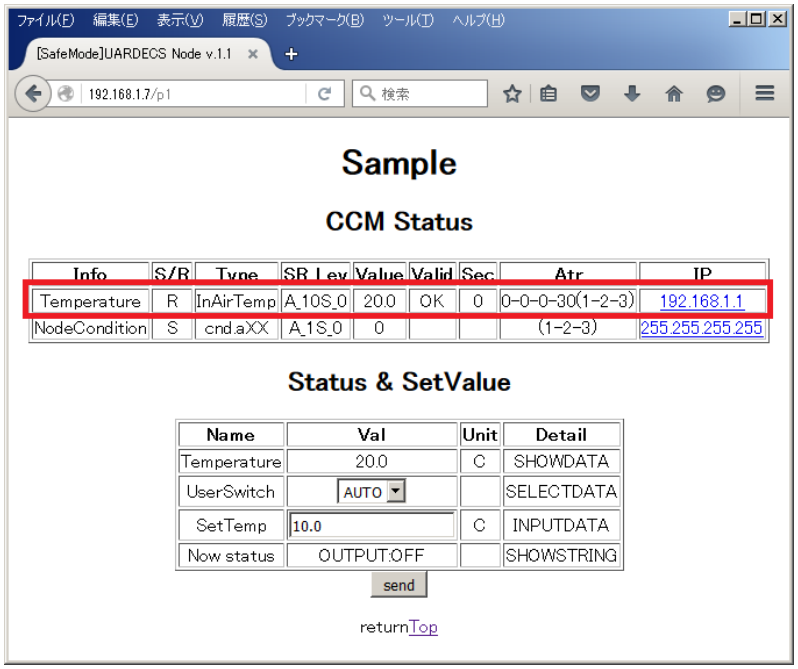
５）ここから先では UECS 用パケット送受信ツール（入手先：http://uecs.org/arduino/uecsrs.html）を使ってノードの動作を検証します。まず、ツールを入手して ZIP ファイルを解凍し、開発用 PC で実行します。初回起動時にネットワークの使用に関する警告が出ることがありますが通信を許可して下さい。



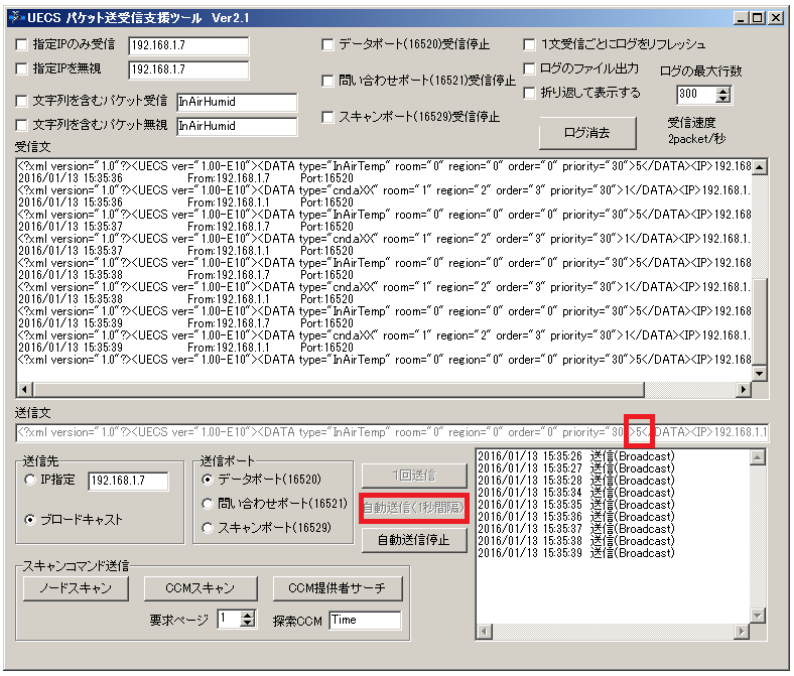
６）PC に接続された Thermostat ノードが正常に動作している場合、図のように送信された CCM が一秒間隔で表示されます。



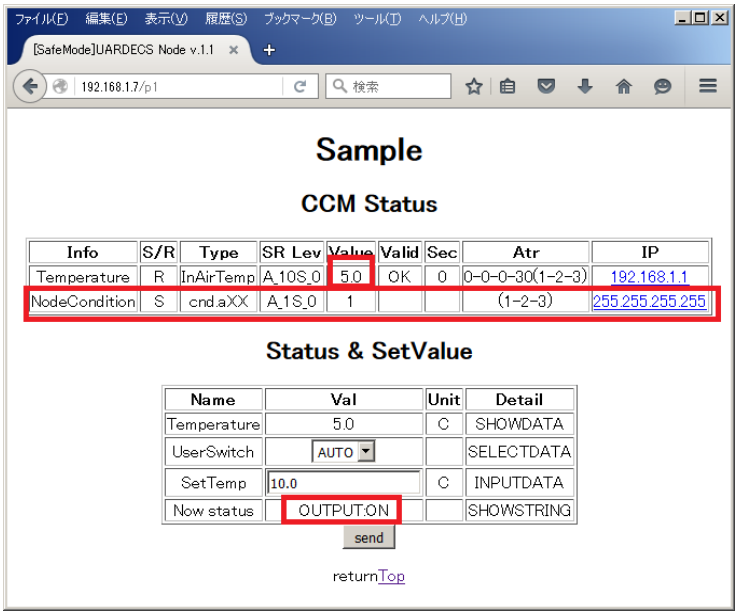
7）試しにパケット送受信ツールから InAirTemp の CCM を送信します。起動時の設定のまま自動送信ボタンを押して下さい。



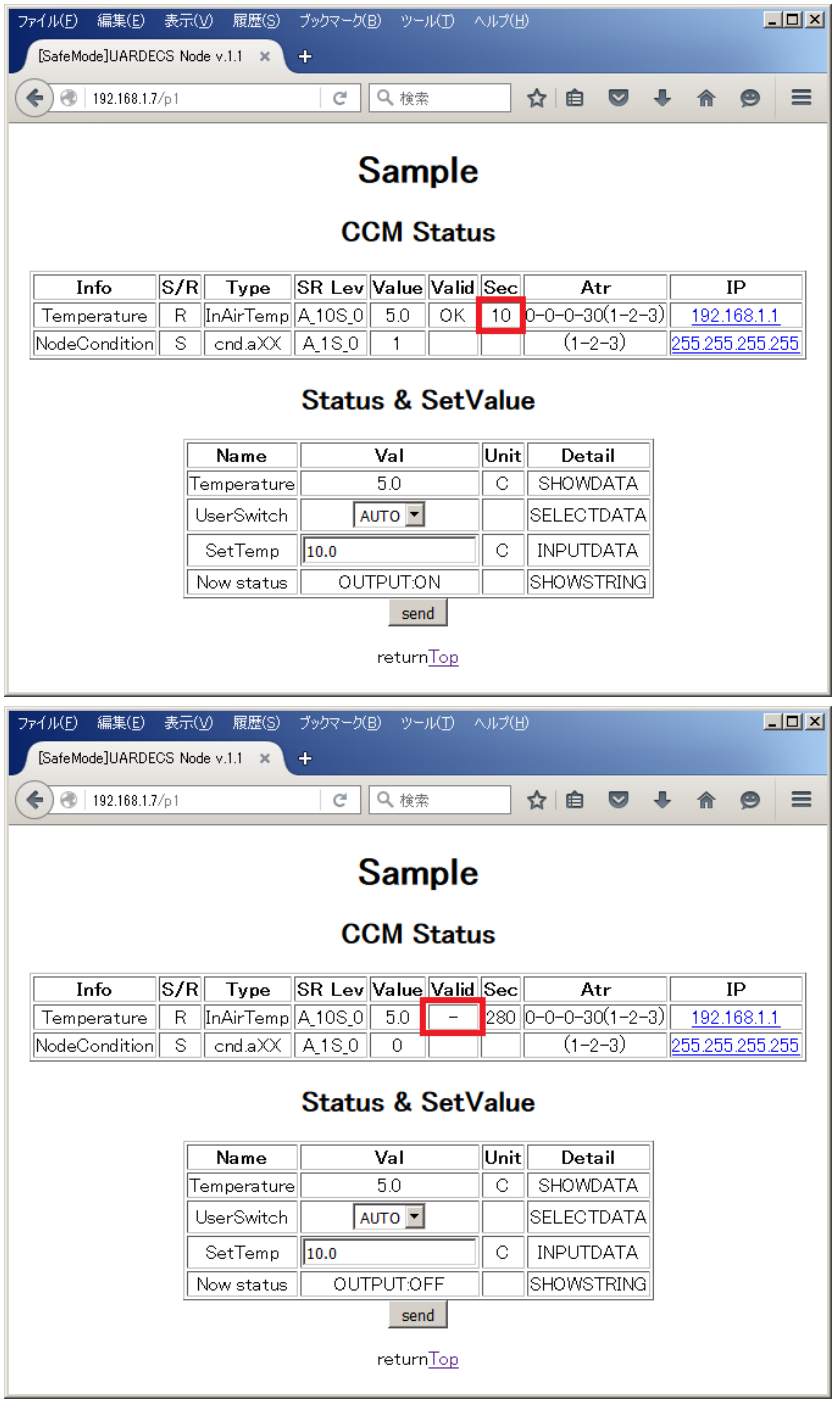
8）Web ブラウザの画面から Thermostat ノードの Node Status に入ると InAirTemp の項目の Valid が OK に変化し、Value にパケット送受信ツールで送った値が表示されます。これで、PC から送られた CCM を正常に受信していることが分かります。



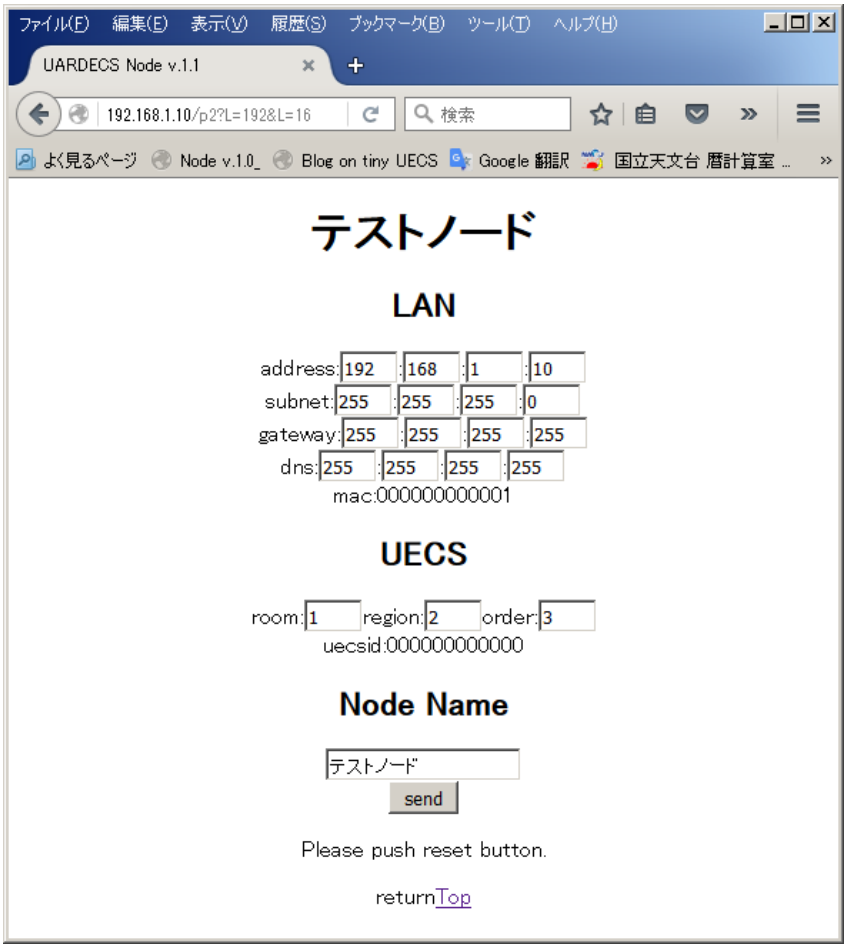
９）送受信ツールで一度パケットの送信を停止した後、InAirTemp の値を 5 に変更してから再度自動送信を行ってみます。



１０）Node Status の InAirTemp の Value が 5.0 に、cnd.aXX の Value が 0 から 1 に変わります。さらに、Now status の所に OUTPUT:ON と表示されます。Thermostat ノードは SetTemp の値(図では 10.0 度)を下回る InAirTemp が入力されると cnd.aXX に 1 を出力するように動作します。



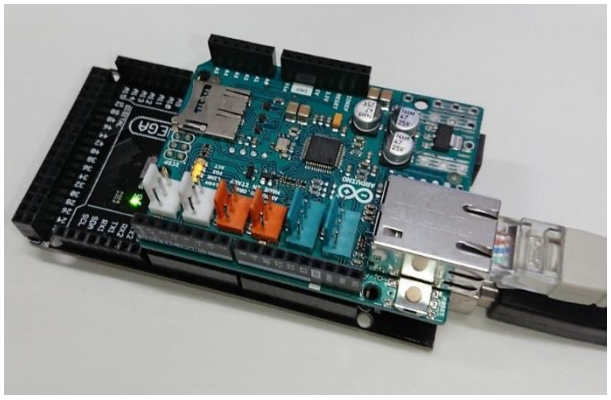
１１）送受信ツールからのパケット送信を停止すると InAirTemp の Sec の部分がカウントを始めます(上図)。InAirTemp の送受信レベルは A_10S_0 に設定されていますが、この設定では受信した値の有効期間は 30 秒です。Sec が 30 を超えると Valid が消え、InAirTemp の値が無効になったことを示します(下図)。このように UARDECS は値の有効期間を自動的に管理します。



１２）Web ブラウザの画面から Thermostat ノードの Network Config にアクセスすると IP アドレスと、UECS に必要な、room、region、order の値(全 CCM がこの値になります)を設定できます。IP アドレスの付け方については、ネット上に多数の文献がありますので参考にして下さい。図では IP アドレスに 192.168.1.10、サブネットマスクに 255.255.255.0 を設定しています。IP アドレス等の設定が書き換わると、リセットを促すメッセージが表示され、リセット後に設定が有効になります。ただし、SafeMode を抜けない限り、IP アドレス設定が有効になることはありません。ノード名には、英数字で 19 文字、日本語 6 文字以内の文字が設定できます。この画面での設定内容は不揮発性メモリに自動的に記録され、電源を切っても値が消えることはありません。デフォルトゲートウェイと DNS サーバも設定可能ですが、UARDECS は現在のバージョンではこの値を活用していません。

８ サンプルプログラムの実行(UARDECS_MEGA)

１）ここから先は UARDECS_MEGA の動作試験を行います。この試験には Arduino MEGA が必要です。Arduino UNO から差し替える場合、６章の４）を参考にボードの設定を変えてください。

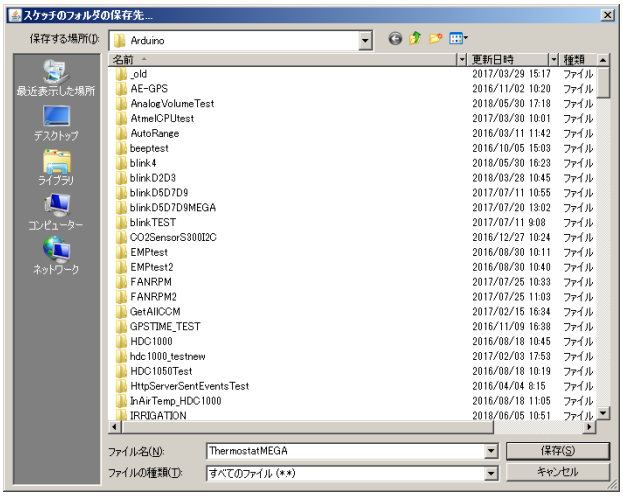


２）UARDECS から UARDECS_MEGA への移行はソースコードのヘッダファイルを書き換えるだけで可能です(UARDECS_MEGA から UARDECS への移行はできないことがあります)。開発用 PC に Arduino MEGA を接続したら７章で使った Thermostat のサンプルプログラムの図の箇所を<Uardecs.h>→<Uardecs_mega.h>に書き換えます。

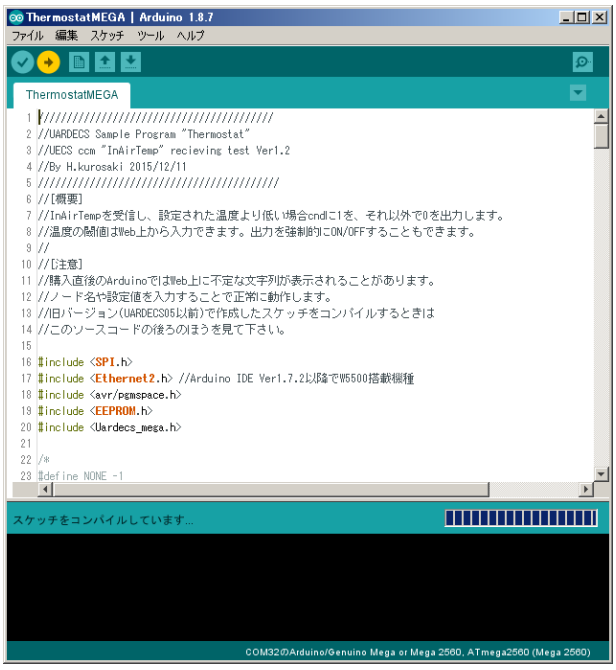
ThermostatMEGA

```
1  ////////////////////////////////////////////
2  //UARDECS Sample Program "Thermostat"
3  //UECS ccm "InAirTemp" recieving test Ver1.2
4  //By H.kurosaki 2015/12/11
5  ////////////////////////////////////////////
6  //[概要]
7  //InAirTempを受信し、設定された温度より低い場合cnd1に1を、それ以外で0を出力します。
8  //温度の閾値はWeb上から入力できます。出力を強制的にON/OFFすることもできます。
9  //
10 //[注意]
11 //購入直後のArduinoではWeb上に不定な文字列が表示されることがあります。
12 //ノード名や設定値を入力することで正常に動作します。
13 //旧バージョン(UARDECS05以前)で作成したスケッチをコンパイルするときは
14 //このソースコードの後ろのほうを見て下さい。
15
16 #include <SPI.h>
17 #include <Ethernet2.h> //Arduino IDE Ver1.7.2以降でW5500搭載機種
18 #include <avr/pgmspace.h>
19 #include <EEPROM.h>
20 #include <Uardecs_mega.h>
21
22 /*
23 #define NONE -1
```

※全く同じソースコードが Uardecs_MEGA のサンプルプログラムの中に入っています

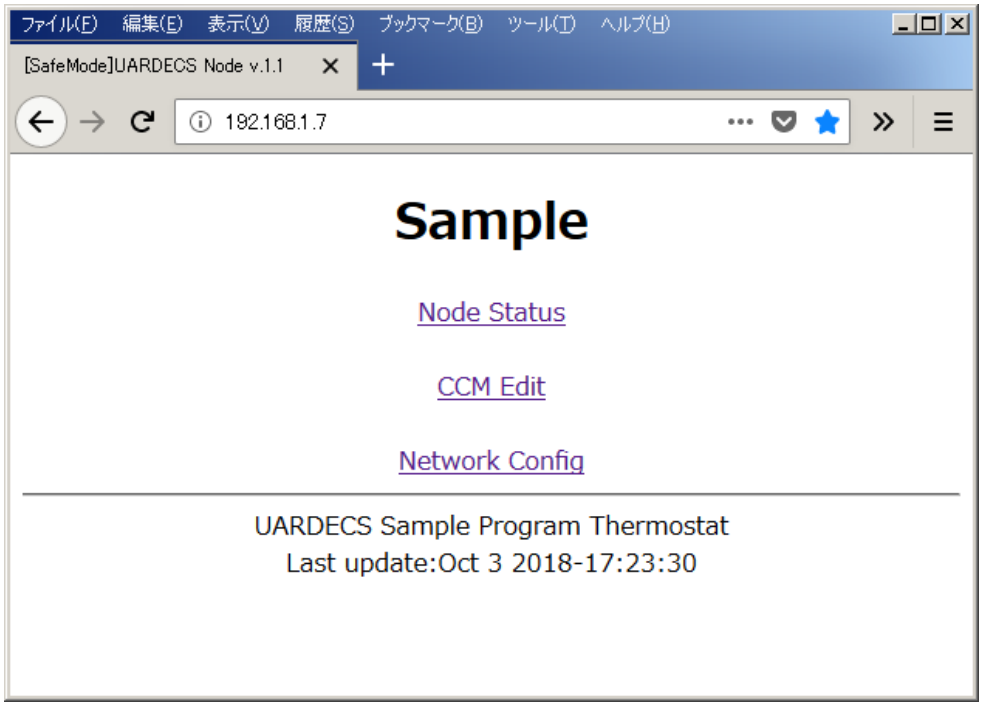


3）サンプルプログラムは書き込み禁止なので、修正したソースコードは別の場所に適当に名前をつけて保存します。



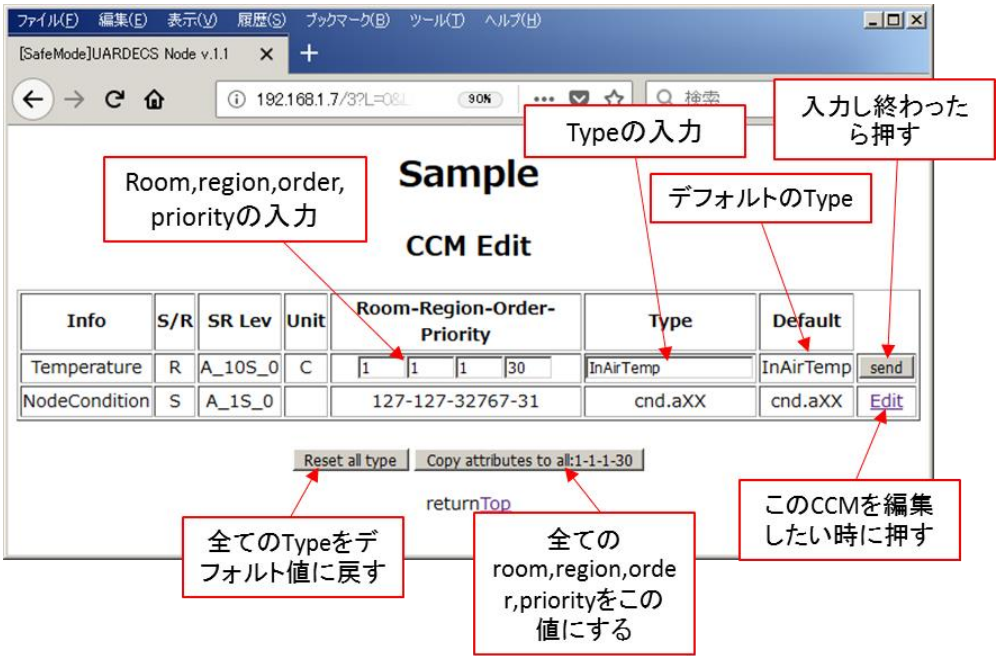
4） IDE 左上の→ボタンをクリックするとコンパイルが始まり、プログラムが自動的に Arduino に書き込まれます。正常に終了すると「マイコンボードへの書き込みが完了しました」と表示されます。

※コンパイルを連打するとタイミングによってはエラーが出ることがあります

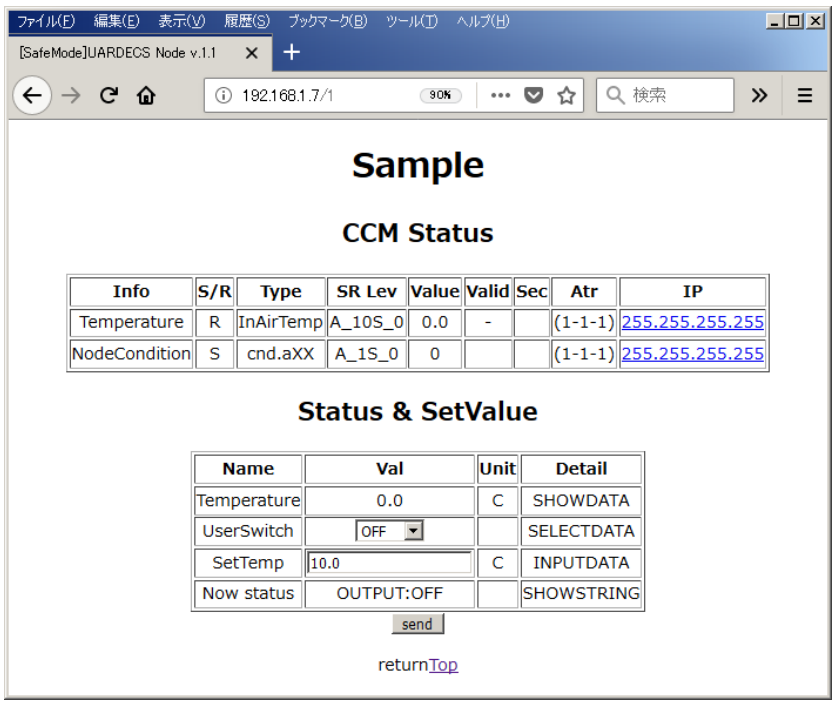


５）Arduino MEGA が LAN ケーブルで接続されていることを確認したら、ブラウザでアクセスします。購入直後の Arduino では自動的に Safemode になり 192.168.1.7 が IP アドレスに設定されています。Safemode の仕様は UARDECS の時と全く同じです。７章の３）を参考にしてください。UARDECS_MEGA では”CCM Edit”という項目が増えているのがわかります。

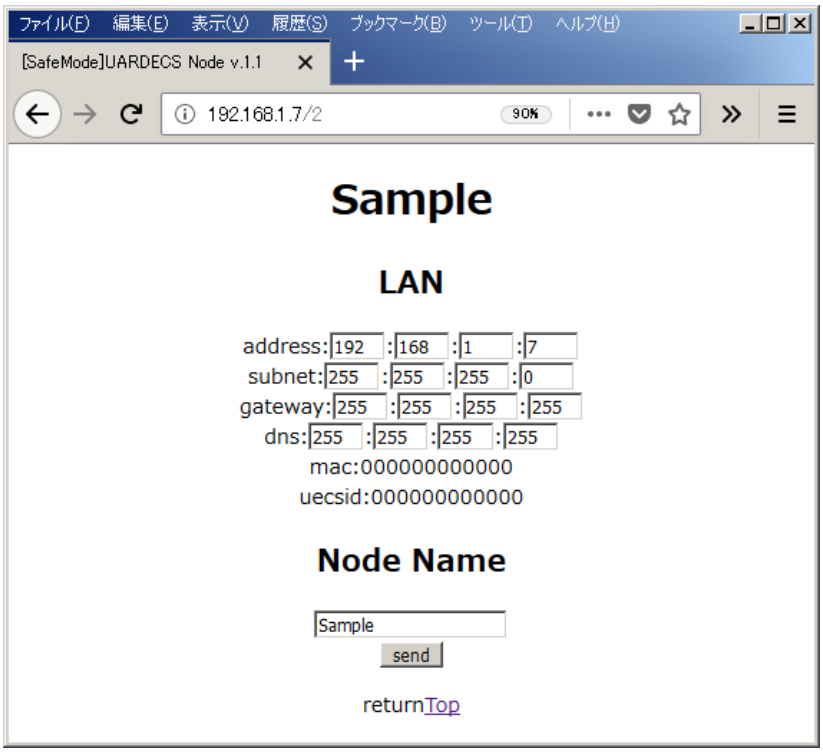
６）最初に CCM Edit に入ります。すると、下のような画面が表示されます。UARDECS_MEGA では全ての CCM の room,region,order,priority をユーザーが CCM ごとにバラバラに設定することができます（UARDECS ではノード単位でしか変更できません）。また、Type の文字列も編集できます。



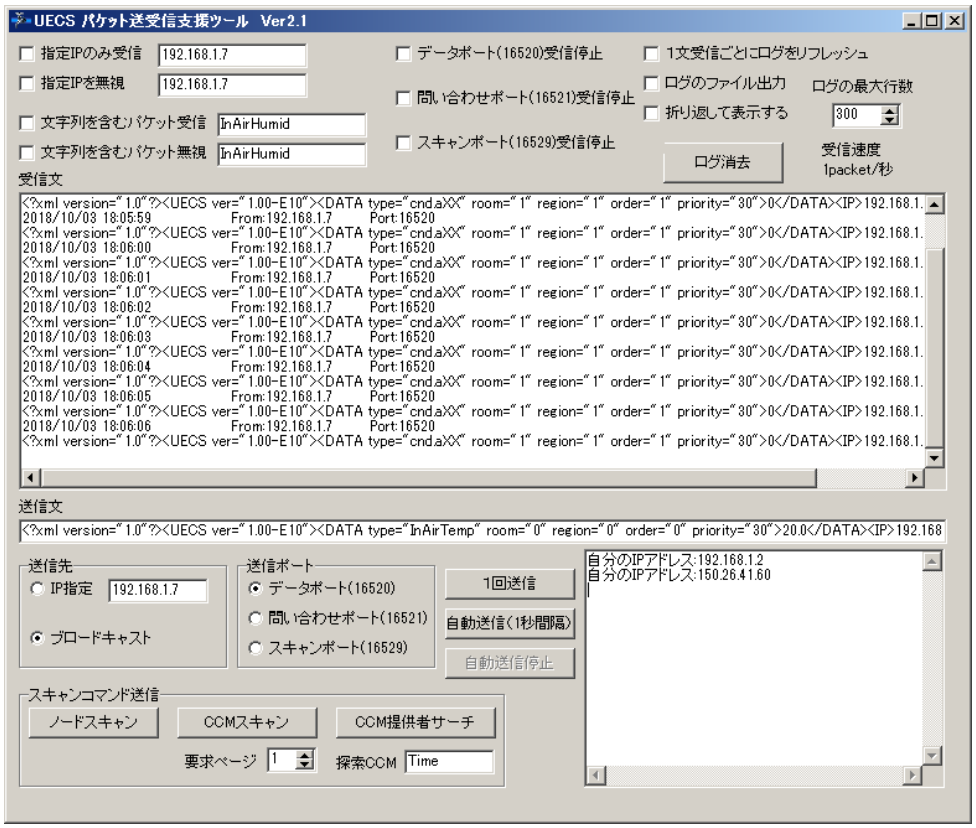
値の修正を行ったら必ず send ボタンを押さないと反映されません。編集する CCM を変更する場合は Edit を押します。下の方にある２つのボタンは、全ての type をデフォルトに戻すボタンと、現在編集中の room,region,order,priority の値を他の全ての CCM にコピーするボタンです。UARDECS_MEGA では新しいプログラムを Arduino に転送したときに Type に不定な文字列が表示されることがあります。その場合、全ての Type をデフォルトに戻すボタンを押して修正してください。



7）Node status 画面は UARDECS と全く同じです。操作方法は７章４）を参照してください。



８）Network Congig 画面も UARDECS とほぼ同じですが、room,region,order の設定は CCM Edit に移ったためここには表示されません。操作方法は７章１２）を参照してください。



9）試しにパケット送受信ツールを起動すると、Arduino が送信する CCM が表示されます。UARDECS_MEGA はユーザーが設定可能な項目が増えていますが、それ以外の部分は UARDECS と全く同じように動作します。

９　スケッチの構成要素解説

ここではUARDECSとUARDECS_MEGAのコンパイルに最低限必要なスケッチを作るための構成要素を解説する。特別な記述がない部分はUARDECSとUARDECS_MEGAで共通である。

１）以下のヘッダファイルを必ずIncludeする必要がある。

```
#include <SPI.h>
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>
```

この<Uardecs.h>の部分はUARDECS_MEGAを利用する場合、<Uardecs_MEGA.h>とする。

２）以下のヘッダファイルは使用するイーサネットシールドの機種によってどちらか一方をIncludeする必要がある。

```
#include <Ethernet2.h>  //W5500搭載機種の場合
#include <Ethernet.h>   //W5100搭載機種の場合
```

３）必ず初期化が必要なグローバル変数

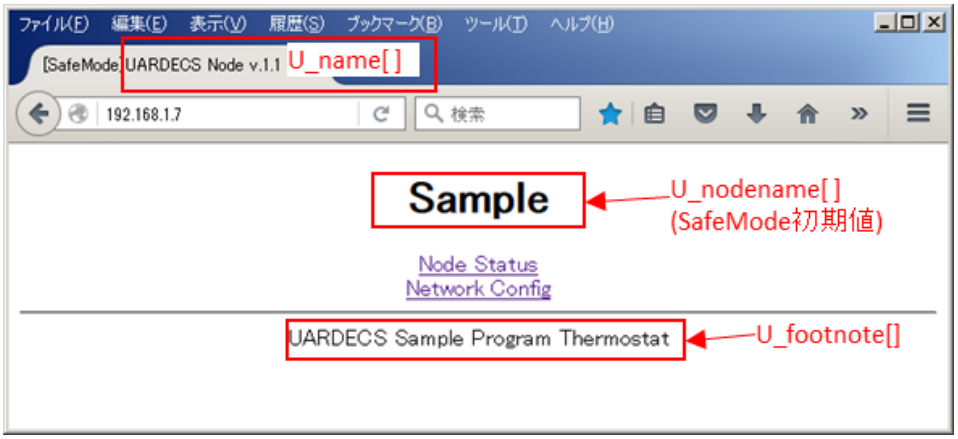
変数の型	変数名	説明
const byte	U_InitPin	SafeModeジャンパーピンの番号を指定。このジャンパーをセットして起動された時、IPアドレスが192.168.1.7、サブネットマスク255.255.255.0に強制的に設定される。購入直後のArduino, IPアドレスを忘れた時に使用する. ここに設定されたピンは入力モードになり自動的にプルアップされる.
const byte	U_InitPin_Sense	SafeModeジャンパーピンの判定条件(HIGHまたはLOW). 起動時にU_InitPinで設定したピンの状態がここに合致するとき, SafeModeとなる.
const char PROGMEM	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). http サーバから出力されるhtmlのタイトルとNODESCANへの応答に使用される.

const char PROGMEM	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)．NODESCAN への応答に使用される．
const char PROGMEM	U_uecsid[ ]	UECS 研究会が発行する ID 番号で 16 進数の 12 桁の数字．半角英数で 12 文字固定．NODESCAN への応答に使用される．試作機用の仮設定は"000000000000" ただし自己責任で)．
const char PROGMEM	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 型の配列(数は 20 で固定)	U_nodename[ ]	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_MAX_CCM	作成する CCM の総数を整数値で．
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．
UECSCCM 構造体の配列	U_ccmList[U_MAX_CCM]	宣言のみで良い．
UECSOriginalAttribute 構造体	U_orgAttribute	宣言のみで良い．

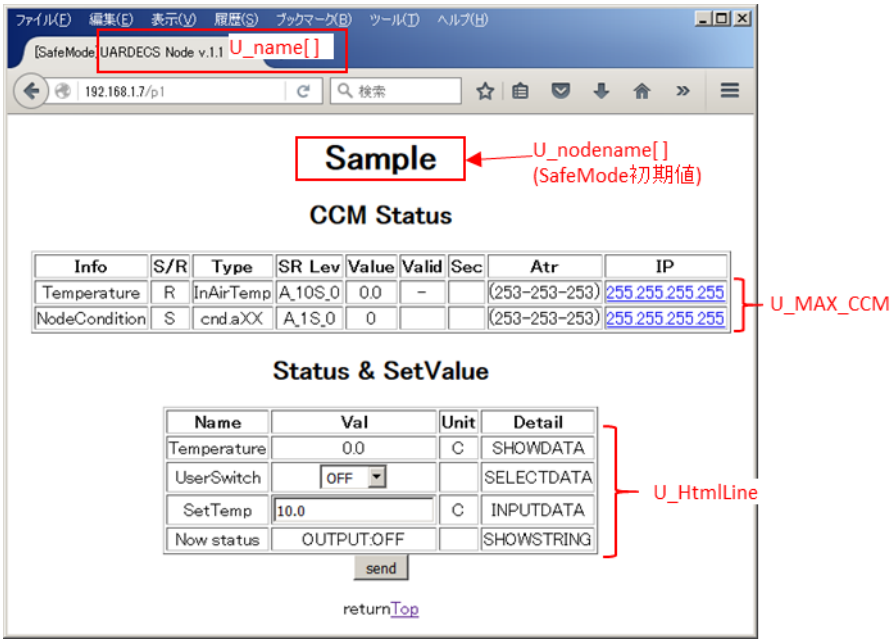
<?xml version="1.0"?><UECS ver="1.00-E10"><NODE>
<NAME>UARDECS Node v.1.1</NAME>
<VENDER>XXXXXXXX Co.</VENDER>
<UECSID>000000000000</UECSID>
<IP>192.168.1.7</IP>
<MAC>112233445566</MAC></NODE></UECS>

U_name[ ]
U_vender[ ]
U_uecsid[ ]
MACアドレス

設定項目の使用箇所①(ノードスキャンへの応答)



設定項目の使用箇所②(Web インターフェースのトップページ)



設定項目の使用箇所③(Web インターフェースの Node Status 画面)

４）必ず作成が必要な関数

関数名	詳細
void UserInit()	Arduino の起動後，ネットワーク設定を読み込んだ後に呼び出される．この関数の後ネットワークが初期化される． この関数内では以下の処理を必ず実行する必要がある． **1）mac アドレスの設定、以下のように記述する** U_orgAttribute.mac[0] = 0x12; U_orgAttribute.mac[1] = 0x34; U_orgAttribute.mac[2] = 0x56; U_orgAttribute.mac[3] = 0x78; U_orgAttribute.mac[4] = 0x9A; U_orgAttribute.mac[5] = 0xBC; Mac アドレスは Ethernet Shield の表面に貼られたシールに書いてある値を入力する．全てのノードに異なるアドレスを付与する必要がある． **2）CCM の作成** [後述する]
void OnWebFormRecieved ()	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．この関数の後，値が不揮発性メモリに格納される．
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.この関数終了後に 16520 番ポートに送信条件を満たした通信文が送信される．
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．
void UserEveryLoop()	システムのタイマカウント，http サーバの処理，UDP16520 番ポートと 16529 番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．
void loop()	この関数内では UECSloop（）関数を呼び出さなくてはならない．必要に応じて処理を記述してもかまわない．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．
void setup()	Arduino の起動後またはリセット後に 1 回だけ呼び出される関数．この関数内では UECSsetup（）関数を呼び出さなくてはならない．必要に応じて処理を記述してもかまわない．主にピンの入出力設定、初期値などを記述する．

５）CCM の作成手順

UECS 通信実用規約 1.00-E10 では全てのノードが必ず１つ以上の CCM を実装しなければならない。ここでは、CCM の初期化方法を示す。

例としてサンプルプログラムの DummyInAirTemp の一部を示し、解説する

手順① 作成したい CCM に通し番号を付ける

この部分には整数をそのまま入力しても良いが、可読性を高めるために DummyInAirTemp スケッチでは以下のように記述している

```
enum {
CCMID_InAirTemp,
CCMID_cnd,
CCMID_dummy,
};
```

このマクロは 2 つの記号定数、CCMID_InAirTemp、CCMID_cnd に通し番号を与えることを示す。CCMID_dummy は作った CCM の総数を数えるのに使うダミーで必ず最後に置く。このマクロによって、

```
InAirTemp       0
CCMID_cnd       1
CCMID_dummy     2
```

のように一意な通し番号が振られる。

以降はこの通し番号を用いて CCM を操作することになる。

手順② CCM の総数を確定し、CCM 格納用の変数を作成する

グローバル変数として以下のように記述する、

```
const int U_MAX_CCM = CCMID_dummy;
UECSCCM U_ccmList[U_MAX_CCM];
```

手順③ CCM を構成する文字列を作成する

必要なのは１つの CCM につき

1, CCM の説明用文字列(Web 表示用、UTF-8 日本語使用可、字数制限なし、タグ文字はそのまま出力される)

2, CCM の type 文字列(InAirTemp.mIC など、半角英字、半角数字、アンダースコア、ピリオドのみ使用可能、3-19 文字)この文字列は UARDECS ではそのまま用いられるが、UARDECS_MEGA ではデフォルト値という扱いになり、後からユーザーが編集した文字列が使われる。

3, CCM の単位を示す文字列(英数のみ 19 文字以下、不要なときは NULL でも良い)

の３種類である。全て const char [] PROGMEM 型のグローバル変数として宣言する。

記述例：

```
// InAirTemp
const char ccmNameTemp[] PROGMEM= "Temperature";
const char ccmTypeTemp[] PROGMEM= "InAirTemp";
const char ccmUnitTemp[] PROGMEM= "C";
// cnd.mIC
const char ccmNameCnd[] PROGMEM= "NodeCondition";
const char ccmTypeCnd[] PROGMEM= "cnd.mIC";
const char ccmUnitCnd[] PROGMEM= ""; //不要なので NULL
```

手順④ UserInit()関数の中で作成する CCM の数だけ UECSsetCCM()関数を実行する

UECSsetCCM 関数の要求する値は以下のとおり

```
UECSsetCCM(bool sender,          //true:送信 CCM false:受信 CCM として宣言する
        int num,                 //手順①で付けた CCM の通し番号
        char* name,      //手順③の CCM の説明用文字列のポインタを与える
        char* type,      //手順③の CCM の type 文字列のポインタを与える
        char* unit,      //手順③の CCM の単位文字列のポインタを与える
        unsigned short priority,//通常は整数値 29、UECS 通信実用規約 1.00-E10 参照
        unsigned char decimal,   //値に小数を使う場合の小数点桁数、0 で整数を示す
        signed char ccmlevel     //送受信の頻度[６) を参照]
        );
```

※UARDECS での UECSsetCCM 関数は UserInit()内での使用を推奨する。UserInit()外での利用は UARDECS_MEGA では不具合の原因になるので、そのような実装は行わないこと。

記述例：

```
void UserInit()
{
UECSsetCCM(true, CCMID_InAirTemp, ccmNameTemp, ccmTypeTemp, ccmUnitTemp, 29, 1, A_1S_0);
UECSsetCCM(true,  CCMID_cnd , ccmNameCnd , ccmTypeCnd , ccmUnitCnd , 29, 0, A_10S_0);
・・・・
```

６）CCM の送受信の頻度（レベル）について

UECS 通信実用規約"1.00-E10"で定める送受信頻度の実装状況は以下のようになっている

(使用するライブラリが UARDECS の場合)

送受信 レベル	送信頻度	受信有効期限	UARDECS での送信	UARDECS での受信
A_1S_0	1 秒	3 秒	**使用可能**	**使用可能**
A_1S_1	1 秒 値が変化した時も送信	3 秒	A_1S_0 と同じ動作	A_1S_0 と同じ動作
A_10S_0	10 秒	30 秒	**使用可能**	**使用可能**
A_10S_1	10 秒 値が変化した時も送信	30 秒	A_10S_0 と同じ動作	A_10S_0 と同じ動作
A_1M_0	60 秒	180 秒	**使用可能**	**使用可能**
A_1M_1	60 秒 値が変化した時も送信	180 秒	A_1M_0 と同じ動作	A_1M_0 と同じ動作
B_0	送信要求があった時	定義しない	未実装	未実装
B_1	送信要求があった時 値が変化した時も送信	定義しない	未実装	未実装
S_1S_0	遠隔操作中:1 秒 遠隔操作しない時:停止	3 秒	**A_1S_0 と同じ動作** **送信停止は手動実装**	**使用可能**
S_1M_0	遠隔操作中:60 秒 遠隔操作しない時:停止	180 秒	**A_1M_0 と同じ動作** **送信停止は手動実装**	**使用可能**

(使用するライブラリが UARDECS_MEGA の場合)

送受信 レベル	送信頻度	受信有効期限	UARDECS での送信	UARDECS での受信
A_1S_0	1 秒	3 秒	**使用可能**	**使用可能**
A_1S_1	1 秒 値が変化した時も送信	3 秒	A_1S_0 と同じ動作	A_1S_0 と同じ動作
A_10S_0	10 秒	30 秒	**使用可能**	**使用可能**
A_10S_1	10 秒 値が変化した時も送信	30 秒	**使用可能**※1	**使用可能**
A_1M_0	60 秒	180 秒	**使用可能**	**使用可能**
A_1M_1	60 秒 値が変化した時も送信	180 秒	**使用可能**※1	**使用可能**
B_0	送信要求があった時	定義しない	未実装	未実装
B_1	送信要求があった時 値が変化した時も送信	定義しない	未実装	未実装
S_1S_0	遠隔操作中:1 秒 遠隔操作しない時:停止	3 秒	**A_1S_0 と同じ動作** **送信停止は手動実装**	**使用可能**
S_1M_0	遠隔操作中:60 秒 遠隔操作しない時:停止	180 秒	**A_1M_0 と同じ動作** **送信停止は手動実装**	**使用可能**

※1 値の変化を検出した場合、1 秒以内に送信される。ただし、値の変化を検出する頻度は 1 秒間隔なので、それ異常高頻度のパケット送信は行われない。

７）CCM の送信

UECSsetCCM()関数で sender を true とし、送信 CCM として登録したものは、設定した送信頻度に従い、一定時間間隔で自動送信される。CCM に値を書き込むときは、以下のように記述する。

記述例：

```
    void UserEverySecond()
        {
        U_ccmList[CCMID_InAirTemp]. value=123;
        }
```

value は long 型の整数である。

UECSsetCCM()関数の decimal 値を 1 以上に設定した場合、value は固定小数点数となり、下位の桁は小数点下の値を表す。例えば上記の記述例において decimal を 1 で宣言した時、CCM として送出される値は 12.3 となる。decimal を 2 で宣言した時は 1.23 となる。

８）CCM の受信

UECSsetCCM()関数で sender を false とし、受信 CCM として登録したものは、設定した受信頻度に従い、値の有効期限が管理される。値が有効かどうかは以下の方法で検出できる。

記述例：

```
    if(U_ccmList[CCMID_InAirTemp].validity==true)
        {
        //値は有効
        }
        else
        {
        //値は無効
        }
```

必ず値の有効/無効を確認してから処理することを勧める。もし、UECSsetCCM()関数で宣言時の decimal が 0 であれば、そのまま long 型の変数として扱える。

記述例：

```
        long ccmTemp= U_ccmList[CCMID_InAirTemp].value;
```

宣言時の decimal が 1 の場合、プログラム内で浮動小数点型に変換するには以下のようにキャストする必要があるかもしれない。
記述例：

```
float ccmTemp=(float)U_ccmList[CCMID_InAirTemp].value/10;
```

※受信 CCM の固定小数点の仕様について
例えば decimal が 2 の状態で外部から以下の CCM 値を受信した場合は次のようになる

受信値	U_ccmList[CCMID].value の値	
100.1	10010	
100.12	10012	
100.123	10012	
100	9999	(変換誤差が発生する)
10	999	(変換誤差が発生する)

９）CCM を送信停止する方法
送信頻度に S_1S_0 などを指定した場合、CCM の送信を一時停止する必要がある。このようなときは、UserEverySecond()関数内に以下のように記述する

```
U_ccmList[CCMID].flagStimeRfirst=false;
```

flagStimeRfirst の値は、この関数を抜けると上書きされるので、flagStimeRfirst に false を書き込んでいる間だけ送信が停止し、この処理を止めると自動的に送信が再開される。この方法を UserEverySecond()関数外で利用することはできない。

１０）U_ccmList[]構造体の詳細仕様

（UARDECS と UARDECS_MEGA に共通の仕様）

UECSCCM 構造体の主要メンバー	変数の型	送信時	受信時
sender	boolean	TRUE	FALSE
name	const char * PROGMEM	文字列ポインタ CCM の説明用文字列(Web 表示用、UTF-8 日本語使用可、字数制限なし、タグ文字はそのまま出力される)	
type	const char * PROGMEM	文字列ポインタ CCM の type 文字列(InAirTemp.mIC など、半角英字、半角数字、アンダースコア、ピリオドのみ使用可能、3-19 文字) UARDECS では CCM の Type として使用される。 UARDECS_MEGA ではユーザーが初期化処理する時以外は利用されない。	
unit	const char * PROGMEM	文字列ポインタ CCM の単位を示す文字列(英数のみ 19 文字以下、不要なときは NULL でも良い)	
ccmLevel	char	記号定数で記述された UECS の通信規約で定められた送受信レベル.	
value	signed long	送信値を格納する	受信値が格納される
decimal	unsigned char	value の小数点以下の桁数	
validity	boolean	未使用	登録した CCM が時間内に取得できているかを示す

recmillis	signed long	未使用	最後に受信してからの経過時間を ms 単位で示す．最大 24 時間分カウントされる(誤差あり)．Web 上に表示されるのは 10 時間まで．flagStimeRfirst が true でない場合、値は保証されない．
address	IPAddress	未使用	CCM 送信元の IP アドレス
flagStimeRfirst	boolean	CCM 送信直前に true になる．false にセットすると送信をキャンセルする． (UserEverySecond()関数内でのみ操作可能)	初めて CCM 受信に成功すると true にセットされる．
attribute[4]	signed short	attribute[3]の priority のみ利用．	受信した CCM に記述された属性値 (baseAttribute と異なることもある) attribute[0]は room attribute[1]は region attribute[2]は order attribute[3]は priority に対応．
baseAttribute[3]	signed short	Web の Network Config 画面で設定したノードの基本属性 baseAttribute[0]は room baseAttribute[1]は region baseAttribute[2]は order	

(UARDECS_MEGA のみで利用可能な変数)

UECSCCM 構造体の主要メンバー	変数の型	送信時	受信時
old_value	signed long	変化があったら送信する CCM で利用される. 過去に送信済みの値を記録する.	未使用
typeStr[20]	char	UARDECS_MEGA ではユーザーが編集した Type 文字列がこの変数に記録されており CCM 送信時に利用される. Web 上から書き換えられる時に同じ文字列が EEPROM 上にも複写され、電源投入時に読み出される.	

送信 CCM の属性値は

U_ccmList[CCMID].baseAttribute[0] →room

U_ccmList[CCMID].baseAttribute[1] →region

U_ccmList[CCMID].baseAttribute[2] →order

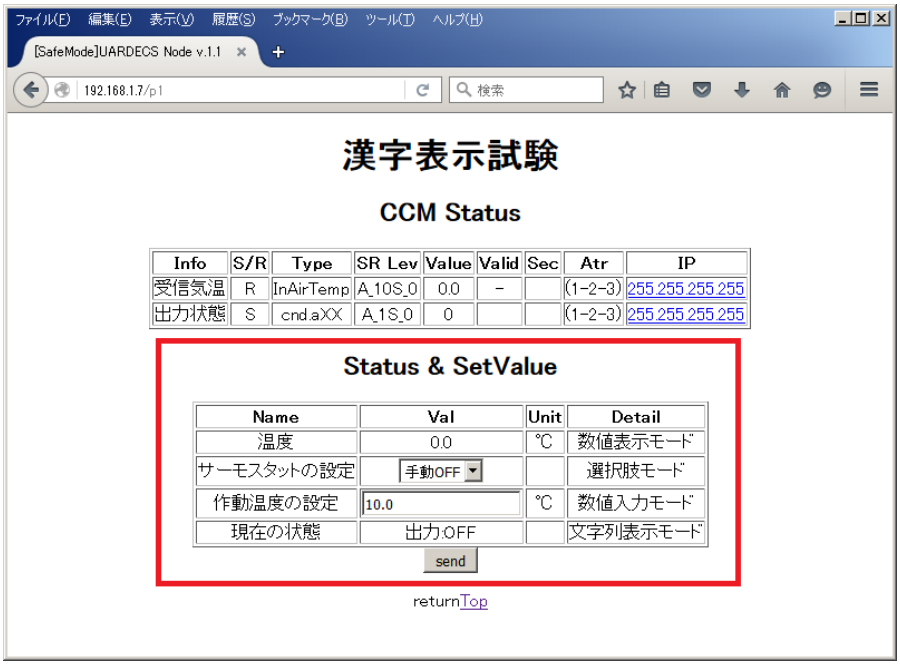
U_ccmList[CCMID].attribute[3]→priority

U_ccmList[CCMID].value→value(decimal により小数点の値を設定)

として出力される。

受信 CCM の場合、UECS 通信実用規約"1.00-E10"で定めた方法で CCM の仕分けを行い、有効と見なされたパケットのみが解釈されてその属性値と送信元の IP アドレスが代入される。

１１）Web インターフェースについて



Web インターフェースはノードにブラウザでアクセスした時、Node Status 画面の下部に表示される設定項目のことである。この設定値は不揮発性メモリに記録され、電源を切っても起動時に自動復帰する。ただし、SafeMode では不揮発性メモリからの値の読み出しは行われず、別途指定した初期値が設定される(不揮発性メモリへの書き込みは可能)。Web インターフェースは以下の４種類の入出力カラムを表示できる。

①UECSSHOWDATA 数値表示カラム(出力用)

指定した数値(整数または固定小数点数)を表示する。

②UECSSHOWSTRING 文字列表示カラム(出力用)

あらかじめ登録した文字列の中から指定した１つの文字列を表示する。

③UECSINPUTDATA 数値入力欄の表示(入力用)

数値入力専用の欄を表示する。整数または固定小数点数の入力が可能である。

④UECSSELECTDATA 選択肢の表示(入力用)

選択肢(ドロップダウンリスト)を表示する。

１２）Web インターフェースの作成手順

手順①

作成する入出力カラムの総数をグローバル変数 HtmlLine に記述する。

記述例：

```
const int U_HtmlLine = 1;
```

手順②

カラムからの入出力用に long 型変数を１つのカラムにつき１つグローバル変数として作成する。

記述例：

```
long webValue;
```

複数のカラムでこの変数を共用した場合、正常な動作は保証できない。

手順③

カラムに表示する素材のうち、カラム名、単位、詳細説明の文字列を準備する。全て const char [] PROGMEM 型のグローバル変数として宣言する。

記述例：

```
const char NAME[] PROGMEM= "Temperature";
const char UNIT[] PROGMEM= "C";
const char NOTE[] PROGMEM= "SHOWDATA";
```

この文字列は UTF-8 で日本語使用可能であり、字数制限は無い。タグはそのまま表示されるので注意すること。

手順④-1　UECSSHOWDATA 数値表示カラム(出力用)の場合

UECSUserHtml 構造体をグローバルとして作成し、素材を入力する。入力する順番は以下のようになる。ダミー値は全て未使用である。UECSSHOWDATA は表示値用変数に格納された値を表示する。小数桁数に 1 以上が指定された場合、固定小数点数として扱う。例えば表示値用変数が 123 で小数桁数が 1 の場合、表示は 12.3 となる。

記述例：

```
const char** DUMMY = NULL; //(const char** 型のダミー値を準備)
struct UECSUserHtml U_html[U_HtmlLine]={{
      NAME,           //カラム名(const char* PROGMEM)
      UECSSHOWDATA,   //カラムの種類(記号定数)
      UNIT,           //単位(const char* PROGMEM)
      NOTE,           //詳細説明(const char* PROGMEM)
      DUMMY,          //ダミー値(const char**)
      0,              //ダミー値(unsigned char)
      &(webValue),    //表示値用変数(long *)
      0,              //初期化値(long) SafeMode で webValue の初期値になる
      0,              //ダミー値(long)
      1               //小数桁数(unsigned char)
      }};
```

手順④-2　UECSSHOWSTRING 文字列表示カラム(出力用)の場合

あらかじめ、素材文字列を準備する必要がある。全て const char [] PROGMEM 型のグローバル変数として宣言する。この文字列は UTF-8 で日本語使用可能であり、字数制限は無い。タグはそのまま表示されるので注意すること。初回起動時に異常な文字列が出力される場合は、SAFEMODE で起動し Node Status 画面で送信ボタンを 1 回押すと修正される。

記述例：

```
const char SHOWSTRING_OFF[] PROGMEM= "OUTPUT:OFF";
const char SHOWSTRING_ON [] PROGMEM= "OUTPUT:ON";
```

次に、素材文字列のポインタを列挙した配列をグローバル変数として準備する。

記述例：

```
const char *stringSHOW[2]={
SHOWSTRING_OFF,
SHOWSTRING_ON,
};
```

次に UECSUserHtml 構造体をグローバルとして作成し、素材を入力する。入力する順番は以下のようになる。ダミー値は全て未使用である。

記述例：

```
struct UECSUserHtml U_html[U_HtmlLine]={
    {
    NAME,              //カラム名(const char* PROGMEM)
    UECSSHOWSTRING,    //カラムの種類(記号定数)
    UNIT,              //単位(const char* PROGMEM)
    NOTE,              //詳細説明(const char* PROGMEM)
    stringSHOW,        //素材文字列のポインタ列挙配列(const char**)
    2,                 //素材文字列の総数(unsigned char)
    &(webValue),       //表示する素材文字列の通し番号 (long *)
    0,                 //初期化値(long) SafeMode で webValue の初期値になる
    0,                 //ダミー値(long)
    0                  //ダミー値(unsigned char)
    }};
```

上記例では webValue =0 の時、Web 上には"OUTPUT:OFF"が表示され、webValue =1 の時、"OUTPUT:ON"が表示される。

手順④-3　UECSINPUTDATA 数値入力欄の表示(入力用)の場合

UECSUserHtml 構造体をグローバルとして作成し、素材を入力する。入力する順番は以下のようになる。ダミー値は全て未使用である。

記述例：

```
const char** DUMMY = NULL; //(const char** 型のダミー値を準備)
struct UECSUserHtml U_html[U_HtmlLine]={{
      NAME,           //カラム名(const char* PROGMEM)
      UECSINPUTDATA,  //カラムの種類(記号定数)
      UNIT,           //単位(const char* PROGMEM)
      NOTE,           //詳細説明(const char* PROGMEM)
      DUMMY,          //ダミー値(const char**)
      0,              //ダミー値(unsigned char)
      &(webValue),    //入力値格納変数(long *)
      -1000,          //最小値(long) SafeMode で webValue の初期値になる
      1000,           //最大値(long)
      1               //小数桁数(unsigned char)
      }};
```

UECSINPUTDATA は数値入力フォームを生成し、入力された値を入力値格納変数に格納する。ノードの http サーバが値を受信すると、OnWebFormRecieved()関数が実行され、この中で入力値格納変数を参照することで更新された値を受け取ることができる。
OnWebFormRecieved()関数実行の後、入力値格納変数の値は自動的に不揮発性メモリに保存される。したがって、受信した値に何らかの操作を加えてから不揮発性メモリに保存させることも可能である。ノードへの電源投入時 UECSsetup()関数実行後に webValue の値は自動復帰する(SafeMode 以外)。小数桁数に 1 以上が指定された場合、入力値は固定小数点数として扱う。最小値、最大値は入力値がこの範囲外にあるとき、自動的に入力値をこの範囲内にクリップする(必ず設定する必要がある)。

例えば小数桁数が 2 の状態で UECSINPUTDATA カラムへの入力値は以下のようになる

入力値	入力値格納変数の値	
100.1	10010	
100.12	10012	
100.123	10012	
100	9999	(変換誤差が発生する)
10	999	(変換誤差が発生する)

手順④-4　UECSSELECTDATA 選択肢の表示(入力用)の場合

あらかじめ、選択肢文字列を準備する必要がある。全て const char [] PROGMEM 型のグローバル変数として宣言する。この文字列は UTF-8 で日本語使用可能であり、字数制限は無い。タグはそのまま表示されるので注意すること。

記述例：

```
const char SELECT0[] PROGMEM= "選択肢 0";
const char SELECT1[] PROGMEM= "選択肢 1";
const char SELECT2[] PROGMEM= "選択肢 2";
const char SELECT3[] PROGMEM= "選択肢 3";
```

次に、選択肢文字列のポインタを列挙した配列をグローバル変数として準備する。

記述例：

```
const char *stringSELECT[4]={
SELECT0,
SELECT1,
SELECT2,
SELECT3,
};
```

次に UECSUserHtml 構造体をグローバルとして作成し、素材を入力する。入力する順番は以下のようになる。ダミー値は全て未使用である。

記述例：

```
struct UECSUserHtml U_html[U_HtmlLine]={
    {
    NAME,              //カラム名(const char* PROGMEM)
    UECSSELECTDATA,    //カラムの種類(記号定数)
    UNIT,              //単位(const char* PROGMEM)
    NOTE,              //詳細説明(const char* PROGMEM)
    stringSELECT,      //選択肢文字列のポインタ列挙配列(const char**)
    4,                 //選択肢文字列の総数(unsigned char)
    &(webValue),       //入力値格納変数(long *)  選ばれた選択肢の番号
    0,                 //初期化値(long) SafeMode で webValue の初期値になる
    0,                 //ダミー値(long)
    0                  //ダミー値(unsigned char)
    }};
```

ノードの http サーバが値を受信すると、OnWebFormRecieved()関数が実行され、この中で入力値格納変数を参照することで更新された値を受け取ることができる。上記例では webValue =0 の時、"選択肢 0"が選ばれている事を示し、webValue =3 の時"選択肢 3"が選ばれている事を示す。OnWebFormRecieved()関数実行の後、入力値格納変数の値は自動的に不揮発性メモリに保存される。したがって、受信した値に何らかの操作を加えてから不揮発性メモリに保存させることも可能である。ノードへの電源投入時 UECSsetup()関数実行後に webValue の値は自動復帰する(SafeMode 以外)。

１３）Web インターフェースについて補足事項

①Web インターフェースが不要なとき

Web インターフェースは CCM 関連の機能とは完全に独立しており、不要な場合は表示しないこともできる。この場合、グローバル変数の初期化部分に以下の２行を記述するだけでよい。

記述例：

```
const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};
```

②文字列素材の使い回し

Web インターフェース用に準備した文字列は複数のカラムで使い回すことでフラッシュメモリの消費量を節約できる。

③作れる入出力カラムの最大数

UARDECS では web サーバに入力される文字列が 299byte を超えた場合、末端を処理できない。URL に付加される文字列は、入力された数値の桁数などで大きく変動するため一概に入出力カラムの最大数を定めることはできないが、32bit の long 型変数が表現できる整数の最大値が 10 桁で、これに符号、少数点、セパレータ文字等を加えると、1 カラム辺り最大 15 byte を消費する。さらに、http サーバへのコマンドや末端のフッタが約 14byte を消費する。したがって、19 個以内に留めたほうが無難である。EEPROM の容量から計算した場合、保持できるカラムの最大数は UNO で約 40 個となるが、その前に web サーバの方がボトルネックになる。

④Web インターフェースに設定された値の書き換え（Ver0.8 以降）
プログラム上から選択肢の位置や入力値の格納変数の内容を書き換える場合、通常は OnWebFormRecieved()内でのみ編集が可能である。それ以外の場所で単純に変数の値だけ書き換えても EEPROM に書き込まれないので正常に反映されない。直接書き換えたい場合は EEPROM の値と整合性を取るため、以下の ChangeWebValue()関数を使用する。

```
bool ChangeWebValue(
        signed long* data,      //書き換えたい変数のアドレス
        signed long value       //書き込む値
        );
```

戻り値：書き込み成功:true　　書き込み失敗:false

記述例：

```
        ChangeWebValue(&(webValue),123);
```

&(webValue)のところには Web インターフェース用に登録された変数のアドレスを記述する。それ以外の変数を記述すると書き込み失敗になる。

この関数は Web インターフェースに設定された最大値、最小値を無視して強制的に値を書き込むので注意すること。また、EEPROM には書き換え回数に寿命があるので高頻度の書き換えは推奨しない（UARDECS 内部では保存された値と比較して差分だけ書き換える方式となっている)。SafeMode では、この関数を用いて値を書き換えても特定の操作時に値が初期化されることがある。

１０　補足事項

１）受信 CCM の特殊オプション(Ver0.7 以降、規約外動作なのでオプション扱い)
起動直後の初期化時に U_orgAttribute.flags|=ATTRFLAG_LOOSELY_VERCHECK;とすると E10 以外のバージョン(初期型ノードなど)のパケットも解釈可能なら受け入れます。

特殊オプションについては使用すると規約外の動作となり、不具合の原因になることもあるので、効果を理解した上で使用して下さい。また、UARDECS のアップデートにより仕様が変わる場合があります。

２）ポインタ列挙配列の要素数を数える方法
選択肢の数などは人為的な計数ミスが発生しやすいですが、UARDECS が独自に実装しているマクロを用いてミスを軽減できます。ただし、メモリの消費量は少し増えます。
例えば

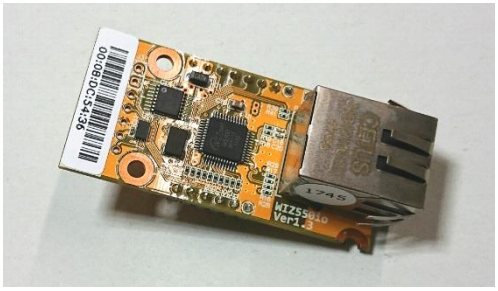
```
const char *stringSELECT[4]={
    SELECT0,
    SELECT1,
    SELECT2,
    SELECT3,
    };
```

という選択肢の列挙配列がある場合、CHOICES(stringSELECT)とすることで要素数を得ることができます。
記述例:

```
struct UECSUserHtml U_html[U_HtmlLine]={
{name1,UECSSELECTDATA,unit1,note1, stringSELECT, CHOICES(stringSELECT),
&(select),0, 0, 0},};
```

３）サードパーティ製のイーサネットシールドの互換モジュール Wiz550io について

これは Arduino のシールドではなく、W5500 の周辺回路のみで構成されたモジュールですが、適切に結線すれば Ethernet Shield2 と同じことができます。



Wiz550io の外観

必要な結線は Arduino→Wiz550io とすると以下の 7 ピンでバス電圧は 5V,3.3V 両用です。

D10→SCSn

(ICSP)MOSI→MOSI

(ICSP)MISO→MISO

(ICSP)SCK→SCLK

RESET→RSTn

3.3V→3V3D

GND→GND

ただし、Arduino 単体では 3.3V の電力供給が不足するおそれがあるので、3.3V のレギュレータを増設する必要があります。

４）Wiz550io 用任意パッチについて

このパッチは現時点で Wiz550io 以外の機種では効果がありません。ドキュメントの"Arduino"フォルダの中の、"libraries/Ethernet2/src"以下の同名のファイルを上書きすることで、MAC アドレスの設定が不要になります。ソースコード内で宣言した MAC アドレスは、W5500 のチップに工場出荷時に内蔵された MAC アドレスで上書きされます。毎回ソースコード内に記述していた MAC アドレスの設定ミスを防ぐのに有効ですが、利用できる機種が限定されるため、任意になっています。純正の Ethernet Shield 2 で使用した場合、メモリの消費量が少し増えるだけでパッチを使用しない時と挙動が変わりません。このパッチは Ethernet2 ライブラリ Ver1.0.3 に適合しますが、他のバージョンでは正常に動作しませんので、Ver1.0.3 をインストールし直してからパッチを当ててください。(ただし、この方法ではバージョンアップの知らせが出たときに指示通りアップデートするとパッチが上書きされて効果が失われることがあります)

１１　よくあるトラブルと対処法

１）Web からの設定が有効にならない、設定した値がリセットすると元に戻っている
SafeMode を抜けて下さい

２）サンプルスケッチをコンパイルして転送しても CCM が送信されない
①本当に「書き込みが完了しました」と表示されている？
プログラムの転送に失敗する場合、(1)文法が間違っておりコンパイルできない
(2)Arduino の種類が間違っている (3)COM ポートの指定が間違っている(4)以前にコンパイルした時の処理が終わっていない、等の問題があることが多いです。

②イーサネットシールドの装着不良をチェック
ピンが正常に刺さっているかチェックして下さい。ピン数の少ない旧仕様のボードなどは刺し位置がずれやすいので注意が必要です。さらに、注意すべき点ですがイーサネットシールドは ICSP と書かれた 6 ピンの端子を使用しています。そのため、Arduino とイーサネットシールドの間にこのピンがないシールドを挟むと機能しなくなることがあります。

③電源容量が足りない
LAN 接続中の Arduino は最低でも約 200mA を消費し、さらに接続したデバイスの消費電力が上乗せされます。電源容量が足りないと起動時に電源がダウンすることがあります。イーサネットコントローラーは 3.3V の電源を多く使用する傾向があります。

③使用するライブラリを間違っている
イーサネットシールドの機種(W5100/W5500 の違い)に適合したライブラリを使用しないとネットワーク関連の機能が使用できなくなります。

３）ノードから PC に CCM は到達しているが、Web サーバにアクセスできない
①ノードと PC のサブネットマスクが食い違っている、IP アドレスが不適切
例えばノードと PC のサブネットマスクが 255.255.255.0 の場合、それぞれに 192.168.1.1～192.168.1.254 の範囲で重複しない IP アドレスを付ける必要があります。

②MAC アドレスが重複している
MAC アドレスが同じノードが複数あると、http の応答が正常にできません。必ず MAC アドレスは全てのノードに異なる値を設定して下さい。

③少し待ってみる
起動から Web ページにアクセス可能になるのに 10 秒ぐらいかかる場合もあります。

4）ノードから PC に CCM は到達しているがノードスキャン、CCM スキャンに応答がない
ノードスキャン、CCM スキャンはブロードキャストではなくユニキャストになるので、2）と同じ項目を点検して下さい。

5）UECS 用パケット送受信ツールでノードにパケットを送っているが反応がない
①CCM の文法が合っていない
CCM の先頭に不正な文字列がある(末端のタグ外の文字列は無視されます)、ヘッダが一致しない、IP タグが足りない等で弾かれることがあります。ただし、IP タグが無い古いバージョンの UECS パケットは 7-1）の設定を行うことで処理できることがあります。

②PC に複数の LAN ポートがある
複数の LAN ポートがある場合、UECS 用パケット送受信ツールで CCM をブロードキャストできるのはどれか 1 つのポートだけになります。不要な LAN ポートのケーブルを抜くなどして無効化するか、IP 指定送信を行って下さい。

6）Web ページのタイトルに[ERR]と表示される
メモリリークしています。特に文字列の字数制限が守られていない時に発生しますが、正規の利用方法で発生する場合、状況などを開発者に連絡いただけると幸いです。

7）CCM のサーチ、CCM 送信リクエストに応答しない
仕様です。16521 ポートへの応答は実装されていません。

8）出力指定したはずのピンから出力が出ない/勝手に信号が出力される
D13 は機種によりブートローダーが起動時に信号を出すことがあります。勝手に信号が出力されるのはシリアル通信や Ethernet Shield などに専有されている可能性があります。回路が短絡しているピンでは、保護回路が働いて Arduino が出力を停止することがあります。

９）Arduino が触れないぐらい発熱する

Arduino の電源回路は入力電圧が高いと発熱が非常に多くなります。特に付属の DC ジャックに DC12V を入力すると触れないぐらいの温度になります。通常は DC9V を入力しますが、それでも発熱が気になる場合、別途 DC5V の電源を用意して USB 端子に入力し、内蔵レギュレータを迂回することで発熱を抑えることができます。

１０）WDT を実装したが、回線が不安定な環境下で http アクセスすると WDT が作動する

Arduino に接続したデバイスによっては通信中に CPU を長時間拘束するものがあります。内蔵 web サーバでのデータ送信中に回線が切断されるとタイムアウトまで少なくとも 32 秒のロック時間が発生します(W5100/W5500 共通)。WDT を使用する場合、このロック時間をフリーズと誤認して WDT が作動してしまうことがあります。この問題を解決するには Arduino IDE の改変が必要です。W5500 ではドキュメントの Arduino フォルダ以下の/libraries/Ethernet2/src/utility の中に socket.cpp というファイルがあり、その中に以下の記述があります（Ethernet2 ライブラリバージョン 1.0.3 で確認)。この do-while 文の内部がタイムアウトまでループする部分ですので、ここに WDT のリセット処理を記述することで不必要な WDT の作動を回避できます。

```
  // if freebuf is available, start.
  do
  {
//←ここに WDT のリセット処理を記述
    freesize = w5500.getTXFreeSize(s);
    status = w5500.readSnSR(s);
    if ((status != SnSR::ESTABLISHED) && (status != SnSR::CLOSE_WAIT))
    {
      ret = 0;
      break;
    }
  }
  while (freesize < ret);
```

W5100 でドキュメントの Arduino フォルダ以下の/libraries/Ethernet/src/utility の中に socket.cpp というファイルがあり、その中に以下の記述があります（Ethernet ライブラリバージョン 2.0.0 で確認)。この do-while 文の内部がタイムアウトまでループする部分ですので、ここに WDT のリセット処理を記述することで不必要な WDT の作動を回避できます。(W5100 はこれ以外にも UARDECS 同封のパッチを当てないと安定して使用できません。)

```
// if freebuf is available, start.
do {
        //←ここに WDT のリセット処理を記述
        SPI.beginTransaction(SPI_ETHERNET_SETTINGS);
        freesize = getSnTX_FSR(s);
        status = W5100.readSnSR(s);
        SPI.endTransaction();
        if ((status != SnSR::ESTABLISHED) && (status !=
SnSR::CLOSE_WAIT)) {
                ret = 0;
                break;
        }
        yield();
} while (freesize < ret);
```

１２　更新履歴

2013.4 Ver0.1

2013.9 Ver0.2

2014.7 Ver0.3

2015.4 Ver0.4

サンプルプログラム、マニュアルを更新しました

U_HtmlLine=0 でフリーズしないように修正

ノード名のメモリリークを修正

Font タグによるメモリリークを修正

U_InitPin_Sense 追加

ArduinoIDEVer1.0.x の UDP 通信のバグを修正

内蔵 wdt を使用しないように変更

2015.7 Ver0.5

W5500 を搭載した新 Arduino 機種に対応

生成されるトップページのフッタが抜けていたのを修正

UDP 通信のバグを再修正

サンプルプログラム追加、マニュアルを更新しました

2015.11 Ver0.6

UserEverySecond(),UserEveryMinute()関数を実装

UserEvery1min()関数の廃止

PROGMEM の記述方法を修正

U_footnoteLetterNumber の定義を廃止

ArduinoIDE Ver1.7.2 以降に暫定対応

メモリ消費量の低減

CCM 受信時のタイムアウト時間が間違っていたのを修正

指定した小数桁数と異なる値を含む CCM が正しく処理できないのを修正

IP アドレスの設定周りを修正

DNS とゲートウェイが正しく設定できないのを修正

NODESCAN の応答時にパケットが途切れるバグを修正

サンプルプログラムの UECSID の桁数が間違っていたのを修正

サンプルプログラムの更新と追加

マニュアルを更新しました

order の最大値が 30000 以上に変更(E10 規約準拠)

Web に文字数の多い選択肢を表示するとメモリリークする問題を修正
大きなパケットを受信した時に発生するメモリリークを修正

2016.1 Ver0.7

マニュアルを大幅更新
ArduinoIDE Ver1.7.8 を標準開発ツールとした
文字列の処理部分を大幅に修正し、フラッシュメモリの消費量を減らした
Web 表示用の文字列生成過程を変更しメモリリークを防止するようにした
Safemode 時に発生する複数のバグを修正
異常に大きな値が記述されたパケットを弾くようにした
約 50 日前に受信したパケットの valid 判定が異常になるのを修正
priority 判定において IP アドレスの優先順位が間違っているのを修正
サンプルプログラムの間違いを修正、サンプルプログラム追加
http サーバの応答が規格に準拠していないのを修正
メモリリークを検出できるようにした（全てを検出できる訳ではありません）
受信 CCM を受信しない状態が 24 時間以上続くと他の CCM のタイムアウト時間の計測が停止するバグを修正

2016.12 Ver0.8

Arduino IDE 1.8.0 対応、警告が出る箇所を修正、パッチのアップデート
Web の Node Status 画面で設定完了後に URL の文字列を初期化するようになった
SafeMode 時に設定されるサブネットマスクの値を修正
不具合の原因になりやすいオプション(ATTRFLAG_ALLOW_ABRIDGE_TYPE)の廃止
Web の Network Config 画面に不要な文字列が出力されていたのを修正
ChangeWebValue()関数を実装
UARDECS アプリケーションガイドとサンプルプログラムを追加

2018/1 Ver1.0

W5100 のサポート終了
UARDECS_MEGA の追加
サンプルプログラムの追加
RTC を積んだ時間を送出するノードで既存の CCM 送信機能を使用すると 1 秒のカウントにズレが生じ不具合の原因となるため、CCM 送信部分をスケッチ側にも記述できるようにスコープを変更した

2018/8 Ver1.0.1

サンプルプログラムの誤字修正

W5100 のサポートを限定的に復活

UARDECS_MEGA でユーザーの入力値のチェック方法を修正

2018/8 Ver1.2.0

マニュアルを大幅更新

W5100 用のパッチを最新版に更新

UARDECS_MEGA でユーザーの入力値のチェック方法を修正(priority の上限修正)

推奨できない実装が行われているサンプルプログラムを削除

１３　旧バージョンとの違いと移行方法

●UARDECS Ver0.5 以前からの違いと移行方法
旧バージョンで開発されたスケッチは一部を書き換える必要があります

(1)インクルードするファイル名が変わりました
CCM.h と EthernetManager.h のインクルードを廃止しました上記ファイルのインクルードを消して Uardecs.h をインクルードして下さい

Arduino IDE Ver1.7.2 以降で動作させる場合
旧 IDE1.0.6 では機種にかかわらず Ethernet.h をインクルードしていましたが、新しい IDE を使う場合、W5100 搭載機種と W5500 搭載機種でインクルードするファイルを書き換える必要があります
W5100 搭載機種(Ethernet Shield R3 など)では Ethernet.h を使用して下さい
W5500 搭載機種(Ethernet Shield 2 など)Ethernet2.h を使用して下さい

(2)以下の関数が実装されました
void UserEverySecond(){}1 秒間隔で実行
void UserEveryMinute(){}1 分間隔で実行
旧バージョンから移行する場合、上記関数が無いとコンパイル時にエラーになるので追加して下さい

(3)以下の関数名が変更されました
void UserEvery1min()の内容は UserEverySecond()に移して旧関数は削除して下さい
void setSendP1Page()の内容は OnWebFormRecieved()に移して旧関数は削除して下さい
※setSendP1Page()では実行前に EEPROM に値が保存されましたが OnWebFormSend()では実行後に保存されます

(4)PROGMEM の付け方が変わりました
[旧バージョン] const char PROGMEM U_name[]
[新バージョン] const char U_name[] PROGMEM
新しい書き方にしないと IDE のバージョンによりコンパイルを通らないことがあります

(5)文字列のポインタ配列にPROGMEMが付けられなくなりました
例：
```
const char *stringSELECT[3] PROGMEM={  <-ダメ
UECSOFF,
UECSON,
UECSAUTO,
};
```
上記の部分は次のようにPROGMEMを外して書いて下さい
```
const char *stringSELECT[3] ={
```
※Webインターフェース用の宣言が影響を受けます

(6)U_footnoteLetterNumberの定義が廃止されました
あってもエラーにはなりませんが、メモリが無駄になるだけです

(7)不要なdefine文が廃止になりました
旧バージョンで宣言していた
″#define NONE -1″から″#define UECSSHOWSTRING 3″までは抹消して下さい

(8)orderの最大値が30000以上になりました(E10規約準拠)
orderのみ上限値が異なります

(9)IPアドレスリセットジャンパーの挙動が変わりました
ジャンパーが有効な場合Webページのタイトルに[SafeMode]の表示が追加されます。この時、IPアドレスは以下の値に強制的に設定されます
IPアドレス:192.168.1.7
サブネット:255.255.255.0
工場出荷時の状態(IPアドレス255.255.255.255)を検出すると
自動的に[SafeMode]に入ります。

(10)漢字表示への対応(Ver0.7以降)
Web用の表示文字列に漢字が使えます。
使い方はサンプルスケッチThermostat_JPのソースコードを参考にして下さい。

(11)起動時間のウェイト調整(Ver0.7以降)

旧バージョンでは問題なかったのにArduino IDE Ver1.7.8以降でコンパイルすると起動時にフリーズするハードウェアがありますが、500msの待ち時間を入れることで電源が安定するまでの時間を確保しました。
(不要な場合、UECSsetup()内のdelay(500);を削除して下さい)

(12)Webアクセスの安定性向上(Ver0.7以降)
Webアクセス中にパケットロスが生じると応答が無くなる問題に対応しました。

(13)Network Config画面の表示追加
MACアドレスとUECSIDが表示されるようになりました。