KashiwaGeeks V0.10

ユーザーズ・ガイド

(注意)

LoRaWAN の基本は unconfirm 通信です。
MAC レイヤーでの送達確認（confirm, LinkCheck）
は必要最小限に留めるのがマナーです。

作成者　山口知昭

作成日　2017.12.13
改訂日　2018.07.13

Application クラス

Application クラスは Arduino の新たなフレームワークで、

1) 省電力のためのスリープ機能
2) ウォッチドックタイマーによるウェイクアップ機能
3) INT0、INT1 割り込み機能
4) タスク実行管理機能

を提供する。

1) #include <KashiwaGeeks.h>

フレームワークが使用できるようになる。

2) void start(void)

Arduino プログラムで使用される setup()に代わる関数。
フレームワークが提供する初期化関数でプログラム起動時に一回だけ実行される。
この関数内に初期設定用のプログラムを記述する。

3) void CosoleBegin(uint32_t baudrate)

Serial.begin()関数に代わる関数でコンソールのボーレイトを設定する。

4) void ConsolePrint(format, ...)

Serial.print() 関数に代わる関数で、可変個の変数を指定されるフォーマットでシリアルポートに出力する。

5) void DebugPrint(format, ...)

ConsolePrint()と同様。

6) void DisableConsole(void)

ConsolePrint()の出力を停止する。

7) void DisableDebug(void)

DebugPrint()の出力を停止する。
DisableDebug() かつ　DisableConsole()　のときは消費電力削減のため UART0 のパワーが オフとなる。

8) void LedOn(void)、LedOff(void)

Arduino の LED を点灯または消灯させる。

9) void sleep(void)

アプリケーションがスリープする直前にこの処理が実行される。
スリープする前に実行したい処理をこの関数内に記述する。

10) void wakeup(void)

アプリケーションがスリープから戻ったときにこの処理が実行される。

11) void EnableInt0(void)

デジタルピン 2 が LOW になった時に割り込みを発生させる。

12) void EnableInt1(void)

デジタルピン 3 が LOW になった時に割り込みを発生させる。

13) void int0D2(void)

デジタルピン 2 が LOW になった時にこの処理が実行される。
実行後はスリープ状態に戻る。

14) void int1D3(void)

デジタルピン 3 が LOW になった時にこの処理が実行される。
実行後はスリープ状態に戻る。

15)　void setWDT(uint8_t sec)

ウォッチドックタイマー値を設定する。　デフォルトは 8 秒となっているが、1, 2, 4, 8 秒のいづれかに設定を変更できる。

16)　TASK_LIST = { TASK(関数、開始時間、繰り返し時間), ... , END_OF_TASK_LIST };

反復して実行したい処理のリストで、このリストで指定した関数を繰り返し時間（分単位）毎に実行される。　開始時間を指定して最初に実行する時間をずらすことができる。

17)　PORT_LIST = { PORT(ポート、関数), ... , END_OF_PORT_LIST };

LoRaWAN からのダウンリンクデータのポートに従って、実行する処理を指定するために使用する。　ADB922S クラスの checkDownLink()メソッドがこれを使用する。

18)　ReRun(関数, uint32_t 開始時間)

開始時間後に指定する関数を実行する。

ADB922S クラスとメソッド

　ADB922S は TLM922S デバイスを使用する LoRaWAN 用の Arduino シールドを意味している。このデバイスは SenseWay、Soracom ともに使用しており、このプログラムはいづれのシールドにも使用できる。

メソッド

1） bool begin(uint32_t baudrate = 9600, LoRaDR dr = DR2, uint8_t retryTx = 1, uint8_t retryJoin = 3);

機能：　ADB922S の初期化を行う。

引数：　uint32_t baudrate：シリアル入出力の速度、9600, 19200, 57600, 115200 が有効
LoRaDR　dr：　最小 DR 値（DR0, DR1, DR2, DR3, DR4, DR5）を設定する
uint8_t retryTx：　送信リトライ回数を設定する。　0 から 255 回が有効
uint8_t retryJoin：　Join のリトライ回数を設定する。

戻り値：正常完了ならば true、　速度設定失敗ならば false。

2） bool join(void);

機能：　LoRaWAN に接続する。

引数：　なし

戻り値：join できれば true。　joinebegin()で設定した retryJoine 数実行しても join できなければ false。

3）bool isJoin(void);

機能：　Join しているか確認する。

引数：　なし

戻り値：join していれば true。　join でしてなければ false。

4）int sendData(uint8_t port, bool echo, const __FlashStringHelper* format, ...);

機能： 文字列データを送信する。 送信後にダウンリンクデータを受信しているか、
8）の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
const __FlashStringHelper* format 送信データフォーマット
… 可変個数の送信データ フォーマットは printf()で使用するものと同じ

戻り値：

LoRa_RC_SUCCESS	正常完了
LoRa_RC_DATA_TOO_LONG	送信データが長すぎる
LoRa_RC_NOT_JOINED	join していない
LoRa_RC_NO_FREE_CH	空きチャンネル無し
LoRa_RC_BUSY	処理中
LoRa_RC_ERROR	その他

5）int sendDataConfirm(uint8_t port, bool echo, const __FlashStringHelper* format, ...);

機能： 文字列データを送達確認付きで送信する。 送信後にダウンリンクデータを受信しているか、 8）の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
const __FlashStringHelper* format 送信データフォーマット
… 可変個数の送信データ フォーマットは printf()で使用するものと同じ

戻り値： sendData() と同じ

6） int sendPayload(uint8_t port, bool echo, Payload*);

機能： ペイロードを送信する。 送信後にダウンリンクデータを受信しているか、
8）の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
Payload* ペイロードのポインター

戻り値： sendData() と同じ

7）int sendPayloadConfirm(uint8_t port, bool echo, Payload*);

機能： ペイロードを送達確認付きで送信する。 送信後にダウンリンクデータを受信しているか、 10）の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
Payload* ペイロードのポインター

戻り値：sendData() と同じ

8） String getDownLinkData(void);

機能： 前回送信時のダウンリンクデータからポートを除く文字データを取得する。
データがなければ空文字列が返される。

引数： なし

戻り値： ポートを除く文字列データ、データ無しは空文字列。

9）Payload* getDownLinkPayload(void);

機能： 前回送信時のダウンリンクデータを Payload クラスとして取得する。

引数： なし

戻り値： 前回の送信時にダウンリンクデータがあればペイロードのポインター、 なければ 0

10） uint8_t getDownLinkPort(void);

機能： 前回送信時のダウンリンクデータからポートを取得する。

引数： なし

戻り値： ポート , 0 は受信データなし。

11） **void checkDownLink(void);**

機能： DownLink データがあればそのポートを取出し、PORT_LINK で指定されたポートと紐付けられたコールバック関数を実行する。

引数： なし

戻り値: なし

12）**bool setADRParams(uint8_t adrAckLimit, uint8_t adrAckDelay);**

機能： ADR に必要なパラメータを設定し、ADR 機能を ON にする。

引数： uint8_t adrAckLimit　ADR_ACK_LIMIT この値を超えたら、ADRReqBit を O
uint8_t adrAckLDelay　ADR_ACK_DELAY この範囲内でネットワークサーバから ACK がなければ、PowerLevel や DR 値が変更される。

戻り値: true は ADR ON。 false は設定失敗。

13）**bool setLinkCheck(void);**

機能： LinkCheckRequest コマンドをFOp ts に設定する

引数： なし

戻り値: true は LinkMargin と NbGateway を取得。 false は応答なし。

14） **bool setTxRetryCount(uint8_t retry);**

機能： データ送信リトライ回数を設定する。 0～255 が有効。

引数： uint8_t retry　送信リトライ設定回数。

戻り値: true は正常完。false は設定失敗。

15） int setADr(LoRaDr dr);

機能：　DR 値を変更する。

引数：　**LoRaDr**　DR0, DR1, DR2, DR3, DR4, DR5。

戻り値：最大ペイロード長。　　-1 は設定失敗

16） void getHwModel(char* model, uint8_t length);

機能：　TLM922S のモデル名を取得する。

引数：　char*model　　モデル名を返すアドレスを指定する。
　　uint8_t length　取得するモデル名の文字数。

戻り値：model に指定文字数分のモデル名が返される。

17）void getVersion(char* version, uint8_t length);

機能：　TLM922S のバージョンを取得する。

引数：　char* version　　バージョンを返すアドレスを指定する。
　　uint8_t length　取得するバージョンの文字数。

戻り値：version に指定文字数分のバージョンがが返される。

18） void getEUI(char* eui, uint8_t length);

機能：　TLM922S のデバイス EUI を取得する。

引数：　char* eui　　デバイス EUI を返すアドレスを指定する。
　　uint8_t length　取得する EUI の文字数。

戻り値：eui に指定文字数分のバージョンがが返される。

19） uint8_t getMaxPayloadSize(void);

機能： setDR()で設定した送信可能ペイロード長を返す。

引数： なし

戻り値：送信可能ペイロード長

20） uint8_t getTxRetryCount(void);

機能： 設定されている送信リトライ回数を取得する。

引数： なし

戻り値：送信リトライ回数

21）void sleep(void);

機能： 無期限のディープスリープする直前に実行する処理。

引数： なし

戻り値：なし

22） void wakeup(void);

機能： 無期限のディープスリープから復帰直後に実行する処理

引数： なし

戻り値：なし

23）void reset(void);

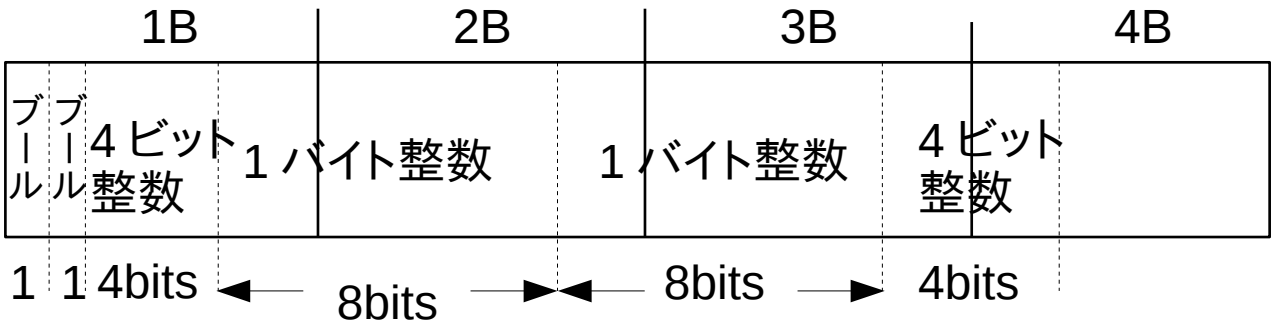
機能： シールドを初期化する。

引数： なし

戻り値：なし

Payload クラスとメソッド

Payload クラスは LoRaWAN のペイロードを表す。 LoRaWAN のペイロードは最長でも 242 バイトである。 通信距離を伸ばすためにペイロードは 11 バイトあるいは 53 バイトにする必要がある。 このため、Payload クラスでは 1 ビットの bool や 4 ビット、1 バイト、2 バイト、4 バイトの整数をバイトの境界を跨いで表現できるようにしたものである。



コーディング例

```
// 4 バイトのペイロードを生成してデータを格納する

Payload   pl(4);

pl.set_bool(true);
pl.set_bool(false);
pl.set_int4((int8_t) -4 );     // -8 ~ 7
pl.set_int8((int8_t) 120);
pl.set_uint8((uint8_t) 250);
pl.set_uint4((uint8_t)15);

bool  b1 = pl.get_bool();
bool b2 = pl.get_bool();
int8_t  i41 = pl.get_int4();
int8_t i81 = get_int8();
uint8_t u81 = get_uint8();
uint8_t u41 = get_uint4();
```

ペイロードからデータを取り出す場合は、格納した順番で取り出していく。

15 文字までの文字列ならば　set_string(String); String get_string(); が使用できる。
15 文字を超える場合は、Payload は使用できない。

データ格納メソッド

```
void  set_bool(bool);
void   set_int4(int8_t);
void  set_int8(int8_t);
void  set_int16(int16_t);
void  set_int32(int32_t);
void  set_float(float);
void  set_uint4(uint8_t);
void  set_uint8(uint8_t);
void  set_uint16(uint16_t);
void  set_uint24(uint32_t);
void  set_uint32(uint32_t);
void  set_string(String);
```

ダータ取出しメソッド

```
bool         get_bool(void);
int8_t       get_int4(void);
int8_t       get_int8(void);
int16_t      get_int16(void);
int32_t      get_int32(void);
float        get_float(void);
uint8_t     get_uint4(void);
uint8_t     get_uint8(void);
uint16_t   get_uint16(void);
uint32_t   get_uint24(void);
uint32_t   get_uint32(void);
String      get_string(void);
```

ADB922S アプリケーション・リファレンスコード

```
#include <KashiwaGeeks.h>
#include <Wire.h>
#include "KGPS.h"

#define ECHO false          <== ADB922S とのシリアル通信内容は表示しない

ADB922S LoRa;               <== ADB922S のインスタンス生成

KGPS gps;                   <== GPS のインスタンス生成
uint8_t portGPS = 12;       <== GPS データの送信先ポート指定
uint8_t portTemp = 13;      <== 温度データの送信先ポート指定

//================================
//       Initialize Device Function
//================================
#define BPS_9600      9600
#define BPS_19200     19200
#define BPS_57600     57600
#define BPS_115200    115200


void start()
{
    /*  Setup console */
    ConsoleBegin(BPS_57600);    <==シリアルポートの通信速度設定

    //DisableDebug();

    /*
     * Enable Interrupt 0 & 1
     */
    EnableInt0();       <==D2 ピンの入力 LOW で割り込み発生させる
    //EnableInt1();   // For ADB922S, CUT the pin3 of the Sheild.

    /*  setup Power save Devices */
    power_adc_disable();     // ADC converter    <==ADC と SPI は使用しないので省電力設定
    power_spi_disable();     // SPI

    /*  setup the LoRaWAN device  */
    if ( LoRa.begin(BPS_19200) == false )    <==ADB922S の通信速度設定
    {
        while(true)                          <==通信速度設定できなければ LED 点滅
        {
            LedOn();
            delay(300);
            LedOff();
            delay(300);
        }
    }

    /* setup the GPS */
    gps.begin(9600, 8, 9);          <==GPS の通信速度とソフトシリアルピンを設定
    ConsolePrint(F("Initilizing GPS\n"));
```

```
    while( !gps.isReady() ){};       <==GPS の初期化と完了を待つ（数分かかる）

    /* setup I2C */
    Wire.begin();              <==I2C の初期化

    /* Set  DR */
    LoRa.setDr(DR3);  // DR0-DR5    <==DR 値設定　（ペイロード長も最大 56B）

    /*  join LoRaWAN */
    LoRa.join();               <==Join 実行
}


//=====================================
// Functions to be executed periodically                    <==タスク宣言
//=====================================
void taskTemp(void)
{
    sendTemp();    <==温度を計測して送信する処理
}
//=====================================
//         Execution interval
//   TASK( function, initial offset, interval by minute )
//=====================================
TASK_LIST = {                                                <==10 分毎に taskTemp()を実行する設定
        TASK(taskTemp, 0, 10),
        END_OF_TASK_LIST
};


//================================
//       INT0 callbaks
//================================
void int0D2(void)                                  <==D2 ピン LOW による割り込みで実行される
{
    DebugPrint(F("********* INT0 *******\n"));
    sendLocation();
}


//================================
//       Power save functions
//================================
void sleep(void)
{
    LoRa.sleep();                    <==Arduino スリープ前にシールドをスリープさせる
    DebugPrint(F("Sleep.\n"));
}
void wakeup(void)
{
    LoRa.wakeup();                   <==Arduino ウエイク直後にシールドをウエイクさせる
    DebugPrint(F("Wakeup.\n"));
}


//================================
// GPS Function
//================================
```

```
void sendLocation(void)          <==GPS を起動して位置を送信する処理
{
    gps.wakeup();
    while( !gps.isReady() ){ };
    Payload* pl = gps.getPayload();       <==位置の Payload を取得する
    if (pl)
    {
        LoRa.sendPayload(portGPS, ECHO, pl);    <==Payload 送信
    }
}

//================================
// I2C ADT7410 Sensor Functions
//================================
void sendTemp(void)                 <==温度を計測して送信する処理
{
  Wire.requestFrom(0x48,2);           <==I2C 読み込み
  uint16_t val = Wire.read() << 8;
  val |= Wire.read();
  val >>= 3;   // convert to 13bit format    <==温度を算出
  int ival = (int)val;
  if ( val & (0x8000 >> 3) )
  {
    ival -= 8192;
  }
  float temp = (float)ival / 16.0;
  char buf[6];
  DebugPrint(F("Temp=%s [C]\n"), dtostrf(temp,3, 2, buf));    <==float を文字列に変換して表示

  Payload pl(LoRa.getMaxPayloadSize());       <==Payload 生成
  pl.set_float(temp);                         <==P 温度（float 値）を Payload に設定
  LoRa.sendPayloadConfirm(portTemp, ECHO, &pl);    <==Payload 送信
}

/*  End of Program  */
```

sendTemp()を sendLocation()に変更して試してみてください。

```
//=======================================
// Functions to be executed periodically
//=======================================
void taskTemp(void)
{
    sendLocation();
}
```