KC4-C-100A

LTE™ Category 4搭載 多機能ゲートウェイ

レシピ言語※をはじめよう

導入編

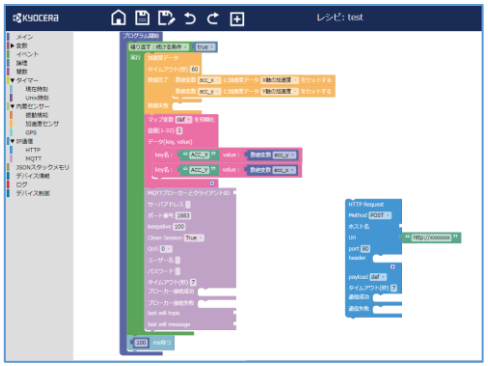
Ver.1.0

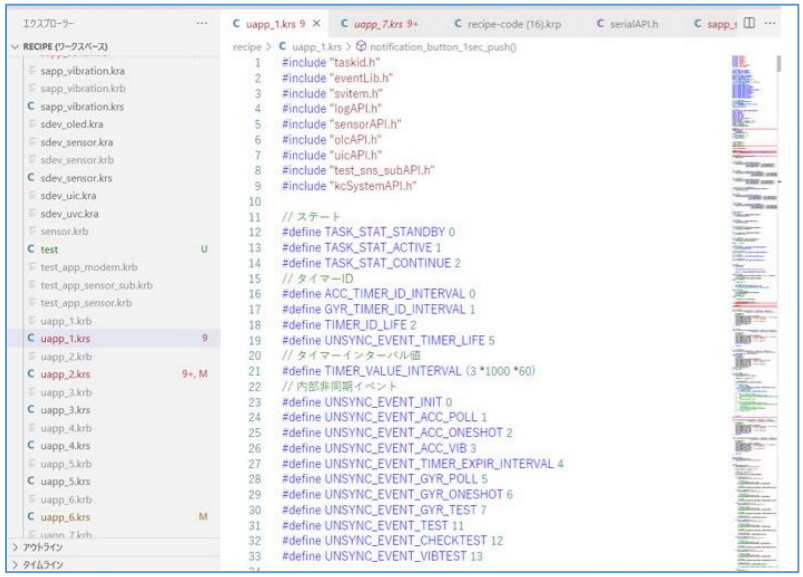
※レシピがコードで組める、C言語に近いコード体系を持つ独自言語です。

レシピ言語とは

レシピ言語はKC4-C-100Aのレシピを直接ソースコードで記述出来るプログラミング言語です。

C言語に近いコード体系を持つ独自言語です。





様々なAPIを準備してありますので、ブロックプログラミング※では制御出来なかった細かなロジックを組むことが出来ます。

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

レシピ言語で出来ること

ここではレシピ言語で実現できる機能の一例を紹介します。

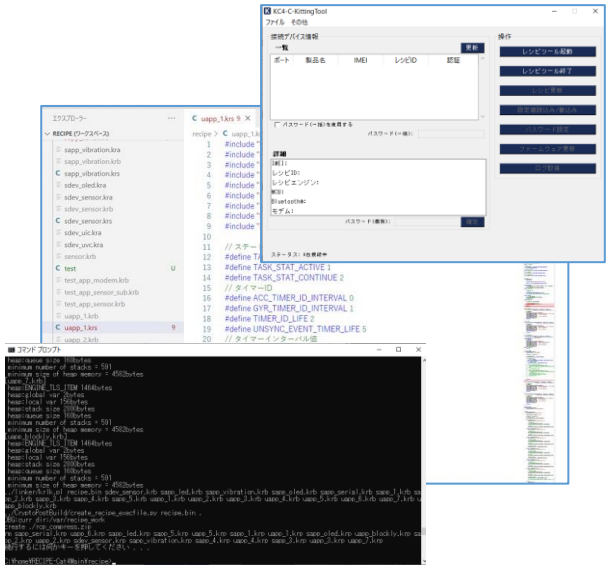
- **マルチタスク**
  最大７つのタスクの並列処理に対応
- **通信機能**
  大手クラウドの認証、サービスに対応
  - Shared Access Signatures
  - TSLクライアント認証
  - ES256認証
  - IoTサービス
  - ストレージサービス
- **Bluetooth®機能**
  送信機能、LongRangeに対応
  - ビーコン送信
  - LongRange送受信
- **シリアル機能**
  バイナリ送信、Modbus通信に対応
  - バイナリデータ送受信
  - Modbusプロトコル対応
  - CAN送受信対応
- **センサー機能**
  詳細制御,データの短周期処理に対応
  - 加速度,ジャイロの詳細制御対応
  - データの10ms周期取得
  - 各種検出アルゴリズム構築対応
- **カメラ機能**
  解像度やフレームレートの詳細設定対応

ブロックプログラムとは違いコードを記述する必要はありますが、機能を制御するための豊富なAPIとサンプルを準備していますので、手軽に本格的なIoTを実現するためのプログラムを効率良く作成が出来ます。

はじめるまでの準備

レシピ言語で開発を行うためには、レシピツールとSDKの準備が必要となります。

ここではレシピ言語がビルド出来る環境の準備を説明します。



- レシピツールのインストール
- SDKのダウンロード、準備
- ビルドの実行
- 機器への書込み

※レシピツールはブロックプログラムと共通ですが、既にインストール済みの場合でも最新のものにアップデートをお願いします。

開発環境構築までの流れ

レシピ言語の開発が開始できるまでの大まかな流れは以下の通りです。

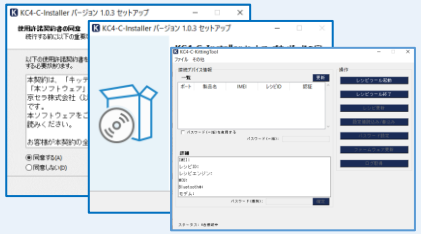
1.レシピツールのインストール

※事前にWSLのインストールが必要です。

ホームページより
インストーラーをダウンロード

KC4-C-Installer_x.x.x.exe

レシピツールをインストール



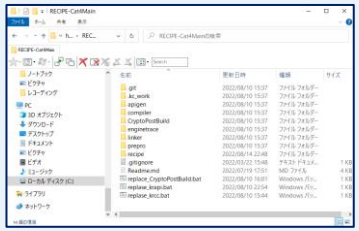
インストール完了（キッティングツールもインストールされます。）

2.SDKの準備

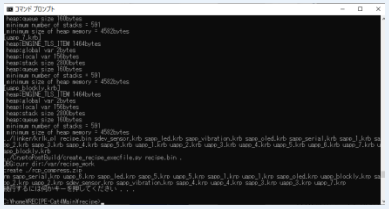
ホームページよりSDKをダウンロード

KC4-C-SDK_1.x.xx.x.zip

SDKを任意のディレクトリに展開



3.ビルド、書込み



コマンドプロンプトからSDKフォルダに
移動し、ビルドコマンドを実行

rcp_compress.zip

実行ファイル作成完了

キッティングツールから機器に書込み

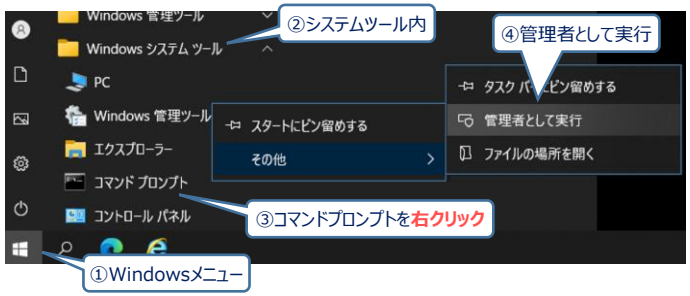
レシピツール、キッティングツールのインストール

レシピツールはLinux®上で動作するため、WindowsにWSL2(Linux®用Windowsサブシステム)をインストール後にインストーラーを実行します。

※既にWSL2がインストールされている場合は
再度のWSLインストールは必要ありません。

1.WSL2のインストール準備

コマンドプロンプトを「管理者として実行」します。



2.WSL2のインストール実施

コマンドプロンプトにて次のコマンドを入力し、
コンピューターを再起動します。

```
wsl --install
```

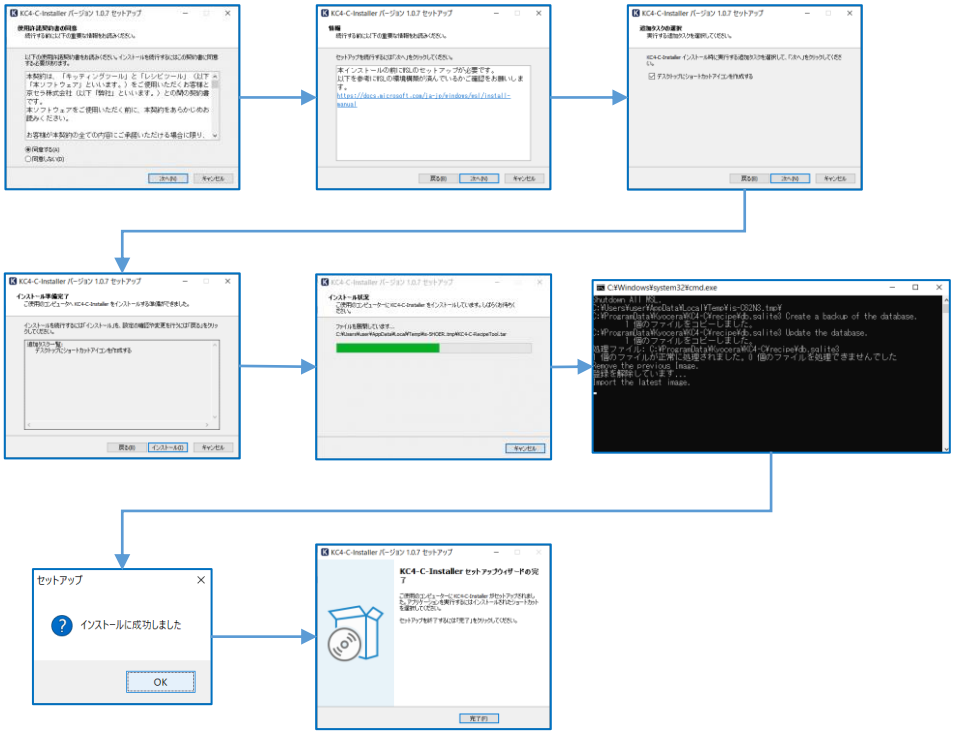
3.レシピツールのインストール

KC4-C-Installerをダウンロードして実行します。

KC4-C-Installer_x.x.x.exe

※インストーラーはダウンロードページより入手出来ます。
https://www.kyocera.co.jp/prdct/telecom/office/iot/development/download/recipe_download.html?start_programing

4.インストーラの画面遷移



SDKの準備（ダウンロードと展開）

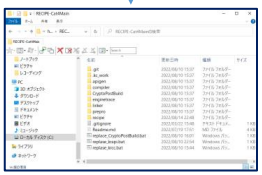
レシピ言語のSDKをダウンロードして任意のディレクトリに展開します。この展開されたSDKのフォルダが作業ディレクトリとなります。

1.SDKをホームページから取得

ZIPファイルをダウンロード

KC4-C-SDK_1.x.xx.x.zip

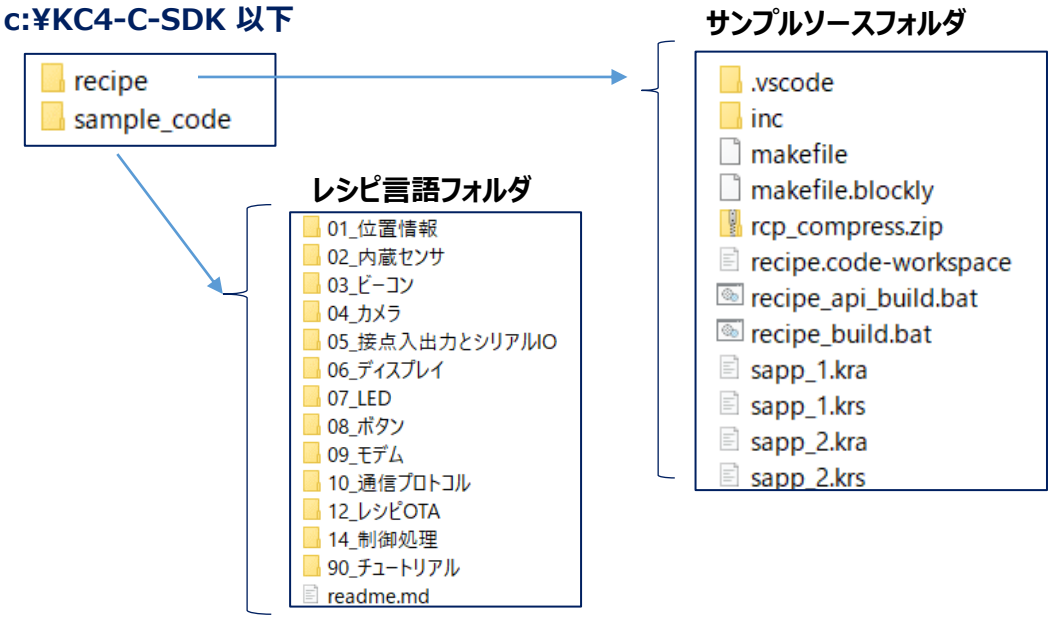
SDKを任意のフォルダに展開



ここでは以下フォルダに展開します

c:¥KC4-C-SDK

2.SDKのフォルダ構造



このレシピ言語ファイルフォルダの中を編集してレシピを作成します。

※インストーラーはダウンロードページより入手出来ます。
https://www.kyocera.co.jp/prdct/telecom/office/iot/development/download/recipe_download.html?start_programing

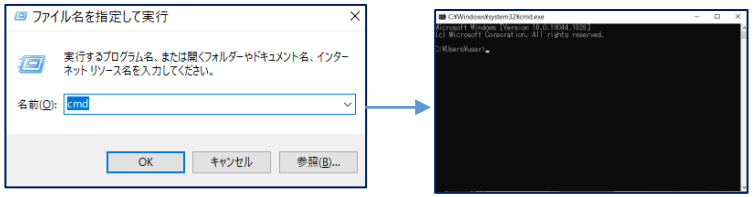
※SDKのフォルダ構成とファイル詳細は次章で説明します。

ビルドの実行

レシピツールとSDKが準備された後は、コマンドラインからレシピ言語のビルドを行います。

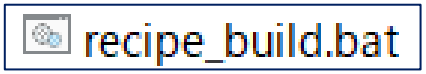
1.コマンドプロンプトの立ち上げ

Windowsキー ＋ Rキーを押して以下ウィンドウから「cmd」と入力。



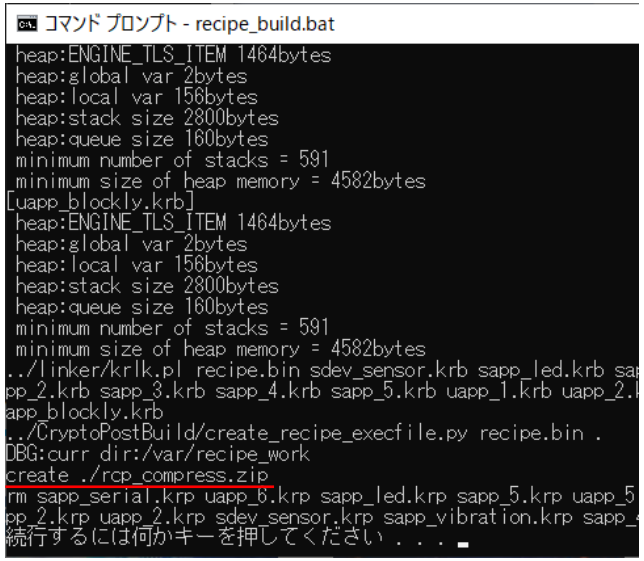
コマンドプロンプト内でフォルダを移動します。
「cd c:¥KC4-C-SDK¥recipe」

「dir」コマンド等でrecipeフォルダの中に、「recipe_build.bat」が存在することを確認します。



2.ビルドの実行

コマンドプロンプト上でbatファイルを実行「recipe_build.bat」



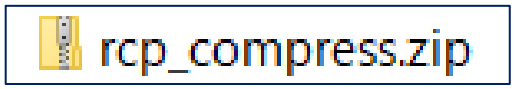
上図の様な出力結果が出ればビルド成功です。

レシピ実行ファイルの書込み

ビルドが成功したら、出来上がったレシピ実行ファイルを機器に書込みます。

1.レシピ実行ファイルの確認

ビルドが成功している場合、recipeフォルダの下に、「rcp_compress.zip」というファイルが出来上がっているので確認してください。



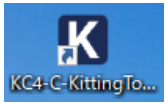
※ビルドした日時になっているか、タイムスタンプも確認してください。ビルドが失敗した場合でも古いファイルが残ります。

2.機器を接続して電源投入

以下のように機器をつなげて電源を入れます。

3.キッティングツールの立ち上げ

デスクトップにある以下のアイコンより実行します。



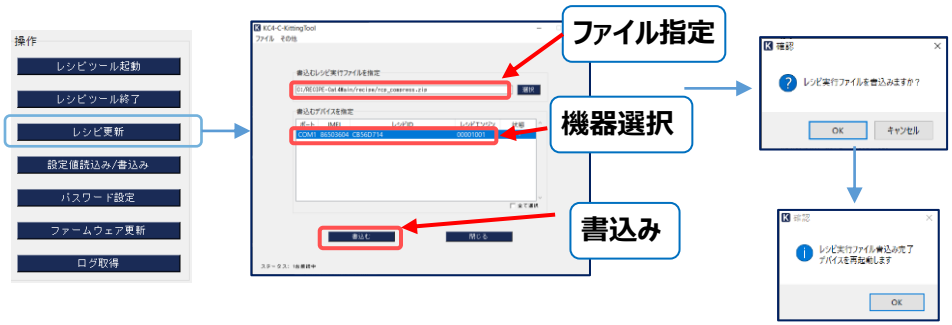
ツール
アイコン

キッティングツール起動

※接続成功した場合ポップアップが表示されます。

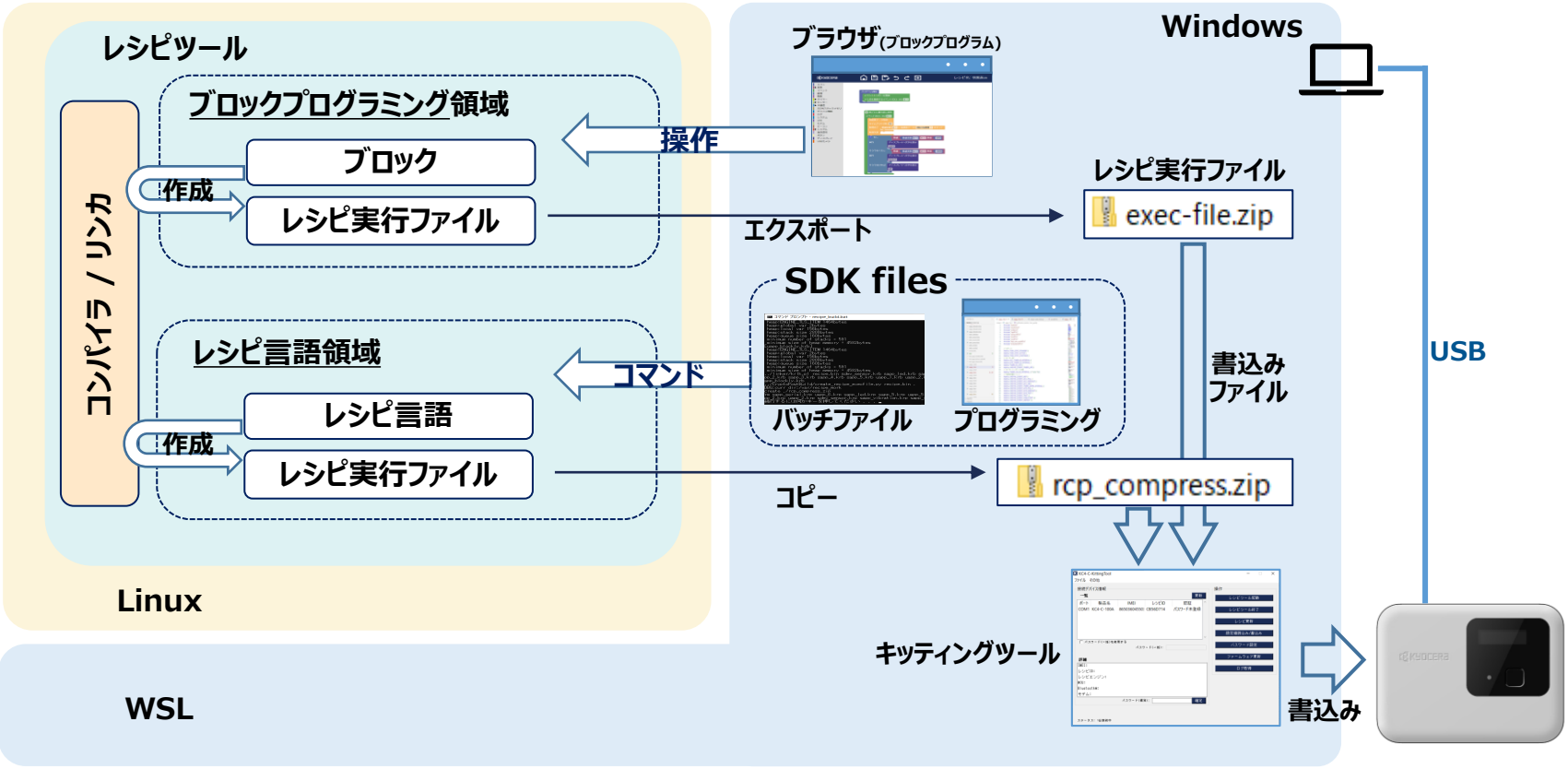
4.レシピ実行ファイルの書込み

レシピ更新メニューから書き込むファイル「rcp_compress.zip」を指定して書込みます。



レシピツールとレシピ言語SDKの関係

順を追ってインストールしてきたレシピツール、レシピ言語SDK、レシピ実行ファイル、キッティングツールの関係性を以下に示します。



はじめての レシピ言語 プログラミング

レシピ言語でプログラミングが出来る環境が整ったところで、まずは簡単なプログラムを動かしてみます。

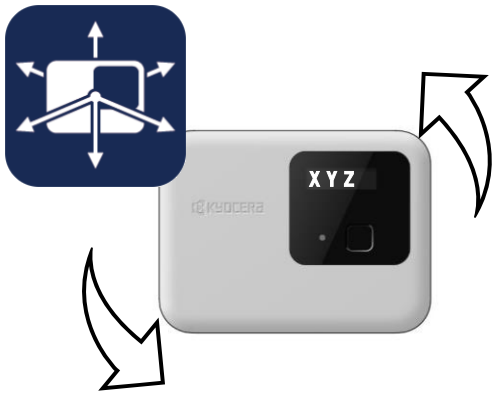
動かすプログラムは、

加速度センサーをつかって、

機器の傾きを検出して、

ディスプレイに表示をする

です。

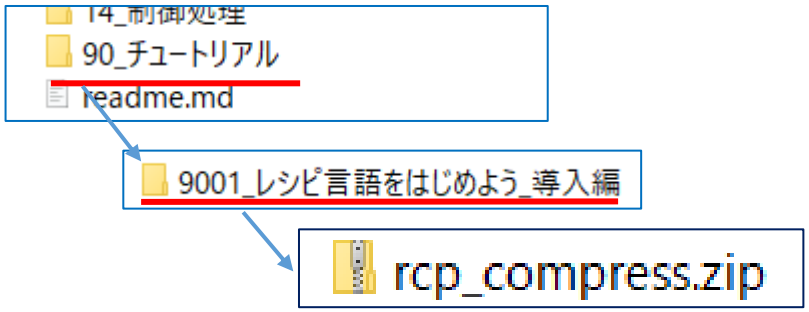


サンプルプログラムで動作確認

サンプルプログラムの中に完成したレシピ実行ファイルがあります。
まず、これを機器に書き込んで動かしてみましょう。

1.レシピ実行ファイルの確認

サンプルプログラムの中から「9001_レシピ言語を始めよう_導入編」のフォルダを探して、その中にあるrcp_compress.zipを確認してください。

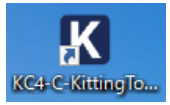


2.機器を接続して電源投入

以下のように機器をつなげて電源を入れます。

3.キッティングツールの立ち上げ

デスクトップにある以下のアイコンより実行します。



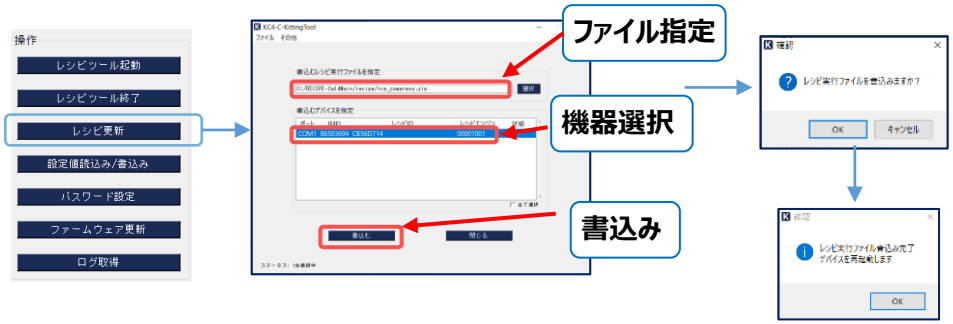
ツール
アイコン

キッティングツール起動

※接続成功した場合ポップアップが表示されます。

4.レシピ実行ファイルの書込み

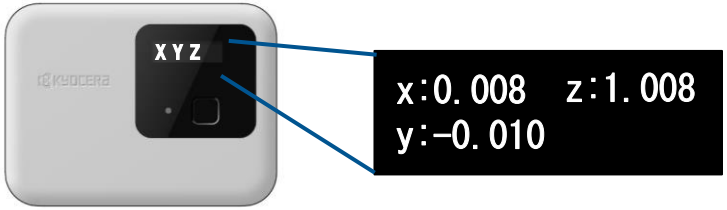
レシピ更新メニューから書き込むファイル
「rcp_compress.zip」を指定して書込みます。



サンプルプログラムで動作確認

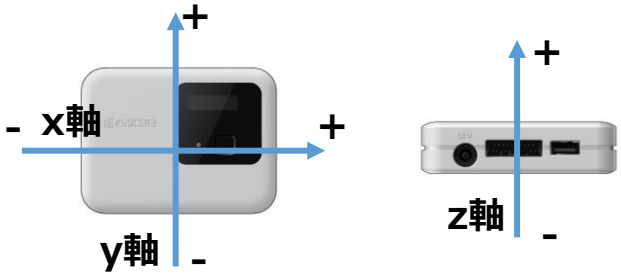
5.機器での動作確認

キッティングツールでレシピを書き終えると、自動的に再起動がされ、レシピが実行されます。



レシピが実行されると、上図の様に画面に加速度の値が、0.5秒毎に表示がされます。

機器と加速度センサーの軸は以下の通りです。
上下左右に傾けながら値を確認してみてください。



※注意：OLEDは焼付きが起きますので長い時間点灯を続けることは避けてください。

6.サンプルプログラムの内容

今回のプログラムは同フォルダに「uapp_1.krs」として置いてあります。今回はこの内容を「ひな形」に加える形で作成して行きます。

※サンプルプログラムの内容の全体は下図の長さです。
少ないコードで動作の記述が出来ます。

```
////////////////////////////////////////////////////////////////////
// include header file //
////////////////////////////////////////////////////////////////////
#ifdef C_SYNTAX
#include "basicInformation.h"
#endif // C_SYNTAX
#include "taskid.h"
#include "eventLib.h"
#include "svitem.h"
#include "logAPI.h"
#include "kcSystemAPI.h"
#include "sensorAPI.h"
#include "olcAPI.h"

////////////////////////////////////////////////////////////////////
// internal define //
////////////////////////////////////////////////////////////////////
// 内部非同期イベント
#define UNSYNC_EVENT_INIT 0
// 加速度センサポーリングイベント
#define UNSYNC_EVENT_ACC_POLL 1

// データ通知インターバルタイム(msec)
#define POLLING_INTERVAL 500

////////////////////////////////////////////////////////////////////
// internal variable //
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////
// event driven task process //
////////////////////////////////////////////////////////////////////
options {
    StackSize = 700;
}

#ifdef C_SYNTAX
void main_loop(){
    WAIT_REQUEST_EVENT_QUEUE
#endif // C_SYNTAX
    rcplib_TASK_InitArg();

    uint16 event_id;
    event_id = rcplib_SV_Read(ENGINE_SVITEM_ID_TASKEVENTITEM_EVENT_ID);

    // 初期化
    if(event_id == UNSYNC_EVENT_INIT) {
        event_init();
    }
    // 加速度センサポーリング時のデータ取得関数
    else if(event_id == UNSYNC_EVENT_ACC_POLL) {
        event_notification_poll(rcplib_TASK_GetArg(0), rcplib_TASK_GetArg(1),
                                rcplib_TASK_GetArg(2),rcplib_TASK_GetArg(3),
                                rcplib_TASK_GetArg(4),rcplib_TASK_GetArg(5));
    }
```

```
    // 強制停止通知
    else if(event_id == EAPI_EVENT_TASK_ENTER_FORCESTOP) {
        KCS_NotifyReadyShutdown();
    }
    // 強制停止解除通知
    else if(event_id == EAPI_EVENT_TASK_RESUME_FORCESTOP) {
    }
    // 機能制限通知
    else if(event_id == EAPI_EVENT_TASK_ENTER_LIMIT) {
    }
    // 機能制限解除通知
    else if(event_id == EAPI_EVENT_TASK_RESUME_LIMIT) {
    }
#ifdef C_SYNTAX
}
#endif // C_SYNTAX

////////////////////////////////////////////////////////////////////
// event function //
////////////////////////////////////////////////////////////////////
// 初期化
func event_init() {

    // Lowパフォーマンスモード、ODR周波数:26Hz
    ACC_SetDeviceMeasure(ACC_LOW_PERF, ACC_ODR3_026Hz);
    // ポーリング実施
    ACC_StartPolling(POLLING_INTERVAL, UNSYNC_EVENT_ACC_POLL);

    OLC_DisplayChar(OLC_CHAR_FORCE,0,0,"              ");
    OLC_DisplayChar(OLC_CHAR_FORCE,18,0,"            ");

    rcplib_LOG_Print("uapp_1 initialize done");

    return(0);
}

// 加速度センサの値を表示する
func event_notification_poll(int16 acc_x, int16 acc_y, int16 acc_z,
                             float acc_f_x, float acc_f_y, float acc_f_z) {
    // 表示する際の文字用の変数
    str ch_x[18];
    str ch_y[18];
    str ch_z[18];

    // センサの値を小数で表示 単位[G]
    rcplib_FORMAT_String(ch_x, "x:%5.3f ", acc_f_x/1000);
    rcplib_FORMAT_String(ch_y, "y:%5.3f ", acc_f_y/1000);
    rcplib_FORMAT_String(ch_z, "z:%5.3f ", acc_f_z/1000);
    rcplib_STR_Cat(ch_x, ch_z);
    OLC_DisplayChar(OLC_CHAR_FORCE,0,0,ch_x);
    OLC_DisplayChar(OLC_CHAR_FORCE,18,0,ch_y);

    return(0);
}
```

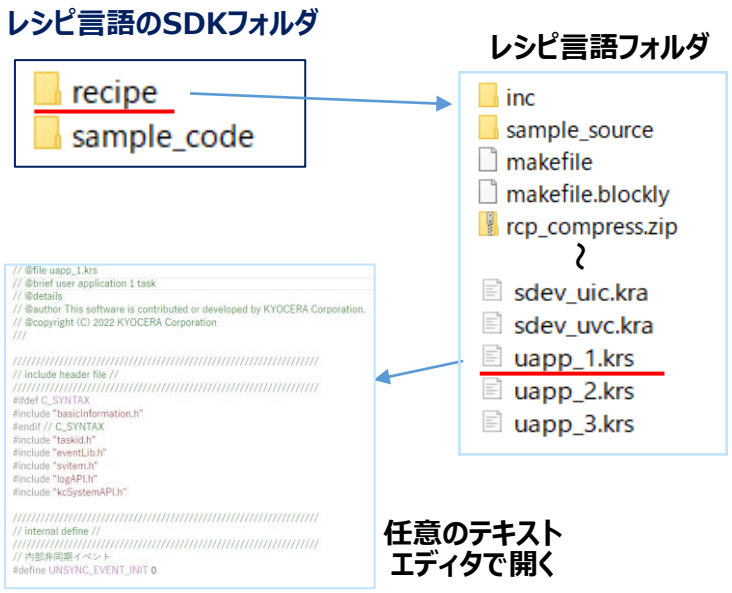
プログラムの作成

ここからは、ひな形をベースに実際のプログラムを作成していきます。

1.ひな形をテキストエディタで開く

プログラムコードを作成する上で、SDKには
コードの「ひな形」を準備してあります。

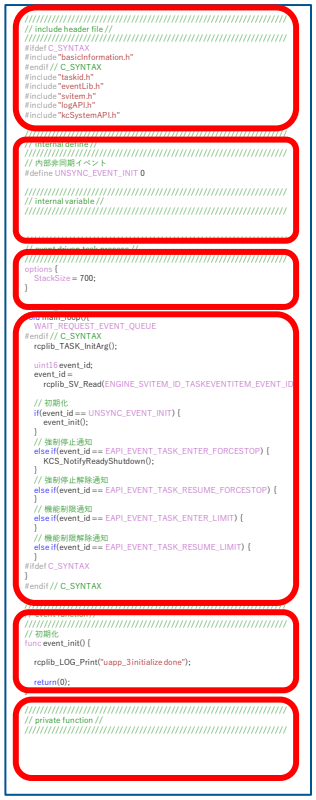
「uapp_1.krs」テキストエディタで開きます。



※レシピ言語では、「*.krs」がソース
コードのファイルとなります。

2.プログラムコードのひな形

プログラムコードのひな形は以下の通り、大枠の配置
場所が決まっています。



■インクルード部
利用するAPIに必要なインクルード
ヘッダを指定。

■定義部
定数のdefine定義やグローバル
変数、タスクの動作設定を定義。

■オプション部
タスクの動作設定等の定義。
※今回はここの説明を省略します。

■イベントループ部
該当イベント毎に処理を分岐する
イベントテーブル。イベントを受信す
ると必ずここに入る。

■タスクのイベント処理
イベントテーブルに対応する関数の
定義/内容を記載。

■追加関数部
イベント毎の関数やその他プライ
ベート関数等の処理部を記載。

プログラムの作成

追加するコードを一つ一つ説明していきます。

1.インクルード部

インクルード部は必要なAPIを利用する為の宣言となります。

今回ひな形から追加するものは、「sensorAPI.h」と「olcAPI.h」になります。

```
////////////////////////////////////
// include header file //
////////////////////////////////////
#ifdef C_SYNTAX
#include "basicInformation.h"
#endif // C_SYNTAX
#include "taskid.h"
#include "eventLib.h"
#include "svitem.h"
#include "logAPI.h"
#include "kcSystemAPI.h“

#include "sensorAPI.h"
#include "olcAPI.h"
```

ひな形に入っている部分

追加部分

加速度センサーやジャイロセンサーを利用する場合は、「sensorAPI.h」が必要となり、ディスプレイへの表示をする場合には、「olcAPI.h」が必要となります。

※詳細はAPI仕様書を確認ください。

2.定義部

定義部ではタスク全体で利用する定数(define)と変数(global)を定義します。

ここではこのタスクが受け取るイベント番号と加速度センサーからデータを受け取る周期を設定しています。

※タスクとイベントは後述します。

```
////////////////////////////////////
// internal define //
////////////////////////////////////
// 内部非同期イベント
#define EVENT_INIT 0

// 加速度センサポーリングイベント
#define EVENT_ACC_POLL 1

// データ通知インターバルタイム(msec)
#define POLLING_INTERVAL 500
```

ひな形に入っている部分

追加部分

上記では500ms周期でデータを受け取る様に定義しています。

※ディスプレイの更新が500msとなるので、この値を小さくしても、実感として早くなった様には見えません。

プログラムの作成

3.イベントループ部①

イベントループ部では、タスクに通知されたイベントを振り分ける処理を定義します。

```
#ifdef C_SYNTAX
void main_loop(){
  WAIT_REQUEST_EVENT_QUEUE
#endif // C_SYNTAX
  rcplib_TASK_InitArg();

  uint16 event_id;
  event_id =
    rcplib_SV_Read(ENGINE_SVITEM_ID_TASKEVENTITEM_EVENT_ID);

  // 初期化
  if(event_id == UNSYNC_EVENT_INIT) {
    event_init();
  }
  // 加速度センサポーリング時のデータ取得関数
  else if(event_id == UNSYNC_EVENT_ACC_POLL) {
    event_notification_poll(
          rcplib_TASK_GetArg(0),
          rcplib_TASK_GetArg(1),
          rcplib_TASK_GetArg(2),
          rcplib_TASK_GetArg(3),
          rcplib_TASK_GetArg(4),
          rcplib_TASK_GetArg(5));
  }
```

ひな形に入っている部分

追加部分

加速度センサーからの通知イベントの振り分け追加します。
ここでは UNSYNC_EVENT_ACC_POLLのイベント番号が通知されてきたら event_notification_poll()をコールします。

※rcplib_SV_Read(),
rcplib_TASK_GetArg()は後述します。

4.イベントループ部②

イベントループ部の続きです。

```
  // 強制停止通知
  else if(event_id == EAPI_EVENT_TASK_ENTER_FORCESTOP) {
    KCS_NotifyReadyShutdown();
  }
  // 強制停止解除通知
  else if(event_id == EAPI_EVENT_TASK_RESUME_FORCESTOP) {
  }
  // 機能制限通知
  else if(event_id == EAPI_EVENT_TASK_ENTER_LIMIT) {
  }
  // 機能制限解除通知
  else if(event_id == EAPI_EVENT_TASK_RESUME_LIMIT) {
  }
#ifdef C_SYNTAX
}
#endif // C_SYNTAX
```

ひな形に入っている部分

ひな形に入っている「強制停止」や「機能制限」の通知はシステムからのアラート通知となります。

main_loop()では処理したいイベントを加えるごとに、else if で分岐を追加していく作業となります。

プログラムの作成

5.タスクイニシャル部

タスクが起動した時に最初に呼ばれる処理となります。

機器が起動するとUNSYNC_EVENT_INITという0番のイベントがシステムから自動的に通知がされて、main_loop()の分岐よりevent_init()関数がコールされます。

```
////////////////////////////////////////////////////
// event function //
////////////////////////////////////////////////////
// 初期化
func event_init() {

   // Lowパフォーマンスモード、ODR周波数:26Hz
   ACC_SetDeviceMeasure(ACC_LOW_PERF, ACC_ODR3_026Hz);
   // ポーリング実施
   ACC_StartPolling(POLLING_INTERVAL, UNSYNC_EVENT_ACC_POLL);

   OLC_DisplayChar(OLC_CHAR_FORCE,0,0,"            ");
   OLC_DisplayChar(OLC_CHAR_FORCE,18,0,"            ");

   rcplib_LOG_Print("uapp_1 initialize done");

   return(0);
}
```

ひな形に入っている部分

追加部分

ひな形に入っている部分(消しても問題ありません)

event_init()の中は基本的にタスクや変数、APIの初期化処理、が入ることになります。ここでは加速度センサーの初期処理とディスプレイのクリアを追加しています。

●使用しているAPIを解説します。

```
// Lowパフォーマンスモード、ODR周波数:26Hz
ACC_SetDeviceMeasure(ACC_LOW_PERF, ACC_ODR3_026Hz);
```

加速度センサーの動きを指定します。
ACC_LOW_PERF：**高いデータ精度を必要としない低消費モードで動作させます。**
ACC_ODR3_026Hz：**ODRは測定データレートで加速度センサーがどの周波数でデータを出力するかを設定します。ここでは26Hzを指定。**
※この指定は加速度センサーのハードウェアに設定する値で、タスクにイベントが通知される周波数ではありません。

```
// ポーリング実施
ACC_StartPolling(POLLING_INTERVAL, UNSYNC_EVENT_ACC_POLL);
```

加速度センサーの動作を開始させます。
POLLING_INTERVAL：**何ms周期でタスクにデータ通知イベントを発行するか設定**
UNSYNC_EVENT_ACC_POLL：**どのイベント番号で通知するかを設定**
※ここで設定したタイミングでmain_loop()に通知が入ります。

```
OLC_DisplayChar(OLC_CHAR_FORCE,0,0,"            ");
OLC_DisplayChar(OLC_CHAR_FORCE,18,0,"            ");
```

ディスプレイへの書込み(画面クリア)を行います。
OLC_CHAR_FORCE：**強制的に上書きすることを指定します。**
※ここでは空白文字を全体に表示する指定をしています。

プログラムの作成

6.追加関数部

今回は一定周期ごとに加速度センサーからのイベント通知が入るように設定をしたため、イベントを実行するための関数を追加します。

```
    // 加速度センサポーリング時のデータ取得関数
    else if(event_id == UNSYNC_EVENT_ACC_POLL) {
        event_notification_poll(
                rcplib_TASK_GetArg(0),
                rcplib_TASK_GetArg(1),
                rcplib_TASK_GetArg(2),
                rcplib_TASK_GetArg(3),
                rcplib_TASK_GetArg(4),
                rcplib_TASK_GetArg(5));
    }
```

追加したイベントループ部

イベントと処理関数は対になります。

```
// 加速度センサの値を表示する
func event_notification_poll(int16 acc_x, int16 acc_y, int16 acc_z,
                float acc_f_x, float acc_f_y, float acc_f_z) {
    // 表示する際の文字用の変数
    str ch_x[18];
    str ch_y[18];
    str ch_z[18];

    // センサの値を小数で表示 単位[G]
        rcplib_FORMAT_String(ch_x, "x:%5.3f ", acc_f_x/1000);
        rcplib_FORMAT_String(ch_y, "y:%5.3f ", acc_f_y/1000);
        rcplib_FORMAT_String(ch_z, "z:%5.3f ", acc_f_z/1000);
        rcplib_STR_Cat(ch_x, ch_z);
        OLC_DisplayChar(OLC_CHAR_FORCE,0,0,ch_x);
        OLC_DisplayChar(OLC_CHAR_FORCE,18,0,ch_y);

    return(0);
}
```

追加した関数部

●使用しているコードを解説します。

```
rcplib_TASK_GetArg(0),
rcplib_TASK_GetArg(1),
・・・・
```

タスク間通信時の引数を取得します。
加速度センサーから通知されるイベントには以下の値が引数として設定されます。
加速度の読み値： x, y, z (-32768～+32767)
重力加速度変換値： x, y, z (-16000～+16000[mG])
上記が引数の0,1,2・・・に設定されている為、GetArg(0), GetArg(1)と順番に読み出すことにより、値が取得出来ます。
※イベントと引数は後述します。

```
// 加速度センサの値を表示する
func event_notification_poll(int16 acc_x, int16 acc_y, int16 acc_z,
                float acc_f_x, float acc_f_y, float acc_f_z) {
```

イベントの実行関数を定義します。
関数はfuncから始まり、関数名(引数・・)という構成になります。
ここでは、加速度センサーの値が引数として定義されてコールされます。

```
    // 表示する際の文字用の変数
    str ch_x[18];
    ・・・
```

文字変数を定義します。
文字変数はstrとC言語に無い型での定義です。※型の詳細は後述します。

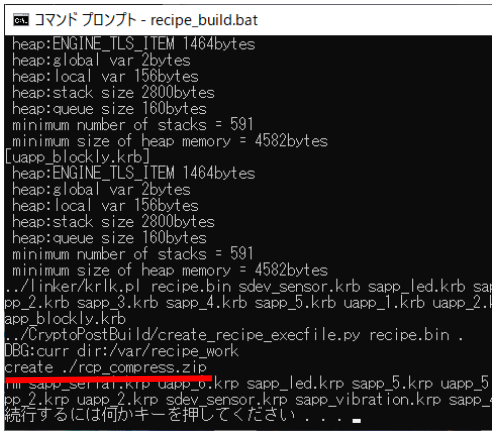
```
// センサの値を小数で表示 単位[G]
rcplib_FORMAT_String(ch_x, "x:%5.3f ", acc_f_x/1000);
・・・
OLC_DisplayChar(OLC_CHAR_FORCE,0,0,ch_x);
```

ディスプレイにセンサー値を表示します。
rcplib_FORMAT_String()のシステムAPIで表示する文字列を整形して、OLC_DisplayChar()のOLED APIでディスプレイに表示をします。

プログラムの作成

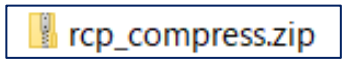
7.ソースファイルのビルド

ソースの追加が終わりましたら、前章の様にコマンドプロンプト上でbatファイルを実行してビルドを行います。
「recipe_build.bat」



上図の様な出力結果が出ればビルド成功です。

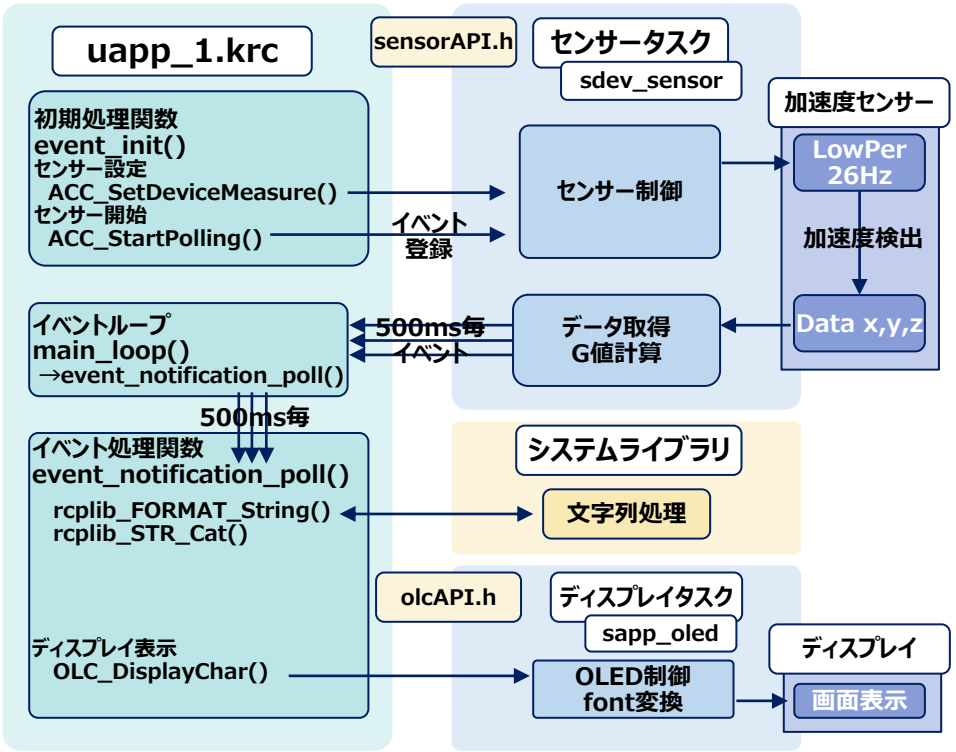
uapp_1.krsと同じディレクトリ上にある



を機器に書き込んで同じ結果となるかを確認してみてください。

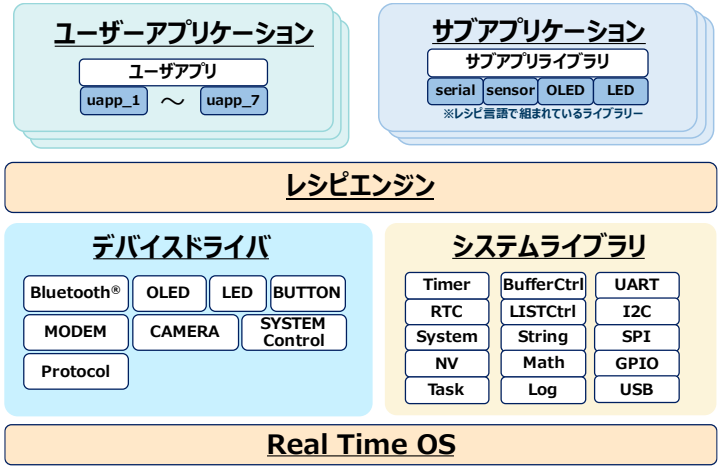
8.全体の動きのイメージ図

今回のプログラムの動きをイメージ図として表現すると以下のようになります。



ソフトウェア構成とSDK、APIの構成

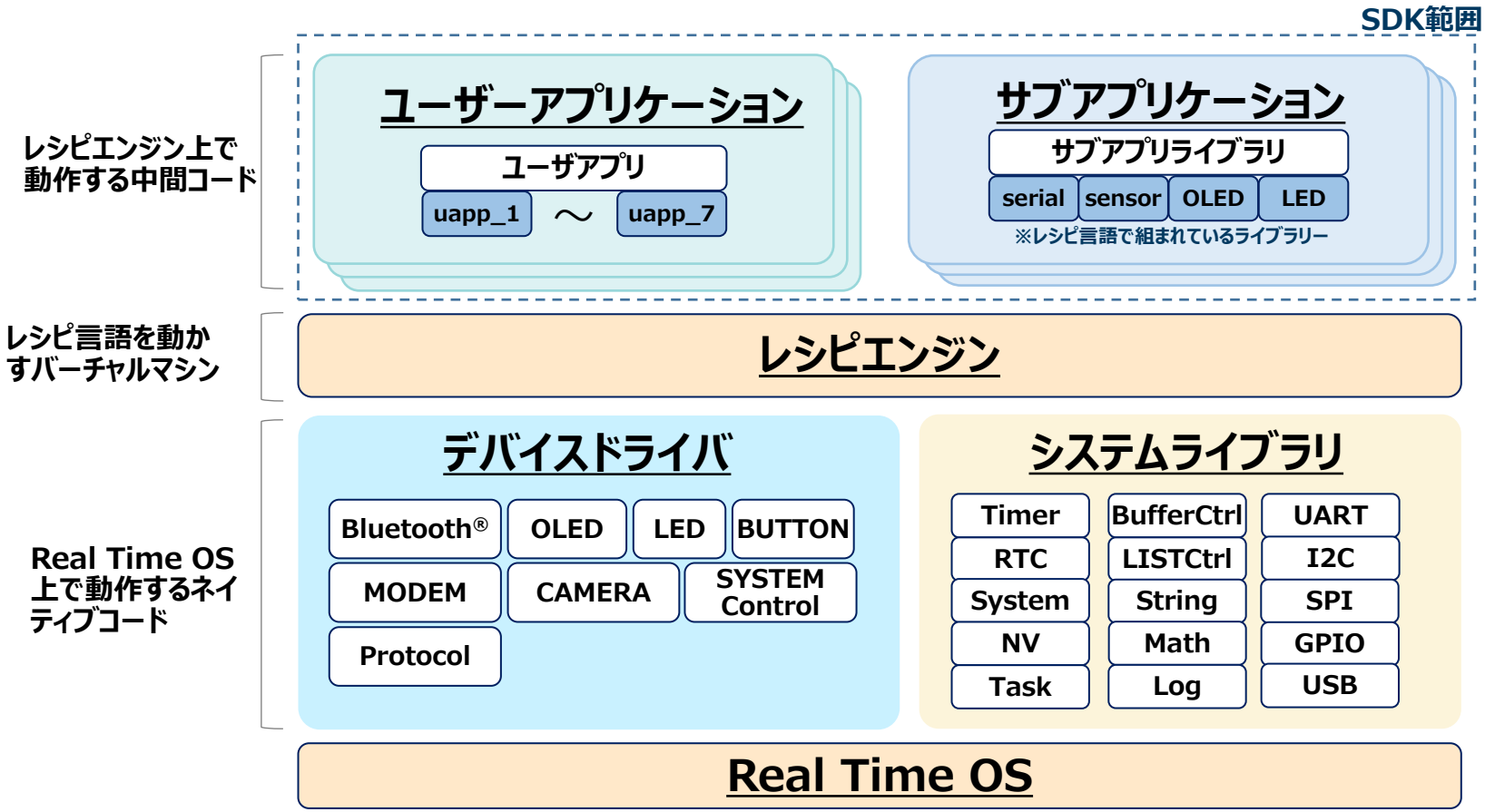
レシピ言語はレシピエンジン上の中間コードとして動作します。ここではソフトウェア構成とSDK、APIの構成について説明します。



- ソフトウェア構成図
- SDKファイルの構成
- レシピ言語ファイルの種類
- サンプルプログラム
- API構成

ソフトウェア構成図

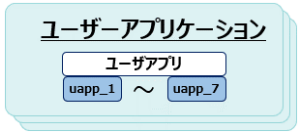
下図にレシピエンジンを含めた本機器のソフトウェア構成図を示します。



※レシピエンジン上のアプリケーション部がSDKとして開発できる範囲となります。

ソフトウェア構成とSDK、APIの構成

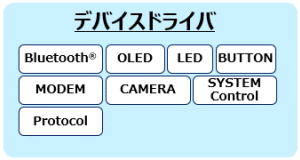
ソフトウェア構成図



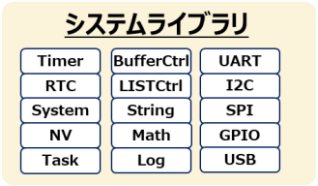
アプリケーション部はレシピ言語で記述され、中間コードにビルドされた後にレシピエンジン上で実行されます。ユーザアプリケーションは基本的にここに記述をします。
uapp1～7までの7つのタスクを使ってアプリケーションを作成出来ます。

サブアプリケーション部はアプリケーション部と同じくレシピ言語で記述されています。アプリケーションを作成してく中で、プロトコルなどのまとまった処理はこちらに定義して、API化をしてユーザーアプリケーション側に提供することが出来ます。

レシピエンジン部は中間コードに変換されたレシピ言語を実行するバーチャルマシンの役割を担います。前述のレシピ言語のビルドは、ここで動作する中間コードを作成する作業となります。



デバイスドライバ部はハードウェア制御や高速な処理が要求されるデバイス制御のAPI提供をしており、ネイティブコードで実行されます。ここはSDKでは提供されずAPIアクセスが提供されるドライバとなります。



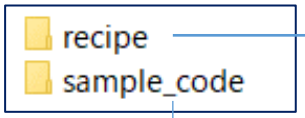
システムライブラリ部はOSで提供される機能やハードウェアを直接制御する機能、システムに関する機能などのライブラリの集合体となります。デバイスドライバと同じくネイティブコードで実行され、それぞれの機能がAPIとして提供されています。

本機器のベースOSはReal Time OSを使用しています。

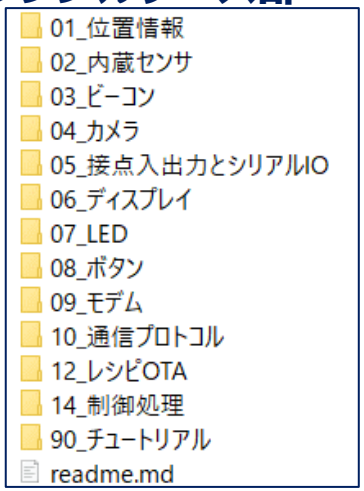
SDKファイルの構成

展開されたSDKは大分類として、「レシピ言語ファイル部」、「サンプルソース」部に分かれます。

フォルダトップ

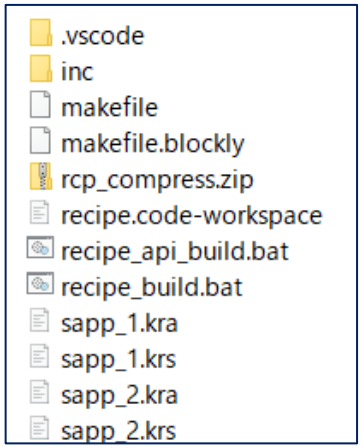


サンプルソース部



参考とする多数のサンプルソースが格納されています。各機能や用途ごとに様々なサンプルコードがありますので、プログラムを効率よく作成することが出来ます。

レシピ言語ファイル部



レシピ言語でプログラムを作成するメインのフォルダとなります。

ここのフォルダのそれぞれのファイルの意味は次ページにて説明します。

レシピ言語ファイルの種類

SDKには多くのファイルがありますが、レシピ言語の記述は対象ファイルが限られています。ここではレシピ言語記述に関連するファイルの種類について説明します。

ファイルの分類/種類

ファイル名	説明
uapp_*.krs	**ユーザアプリケーション**ファイル。 1～7の種類があり、それぞれが別なタスクとして稼働します。 レシピの作成には基本的にこのファイルを編集します。 サンプルコードはuapp_1.krsで組まれているものが多く、置き換えてビルドすることで動きが確認出来ます。
sapp_*.krs	**サブアプリケーション**ファイル。 1～5,led,oled,serial,vibrationがあり、uapp側にAPIを提供出来る仕組みを持つタスクです。 プロトコルなど機能をアプリケーション上で組み、API化してuapp側から使用する使い方をします。
sdev_*.krs	サブアプリケーション内のセンサー処理 sensorは優先度の高い処理としてサブアプリ内にデバイスドライバとして定義。
recipe_build.bat	レシピ言語ビルドを行うバッチファイル。Linux上のレシピツールを操作してレシピ実行ファイルを作成します。
makefile	ビルドを行うポリシーを定義するファイルです。
¥inc	API等を定義したヘッダファイルが格納されているディレクトリです。

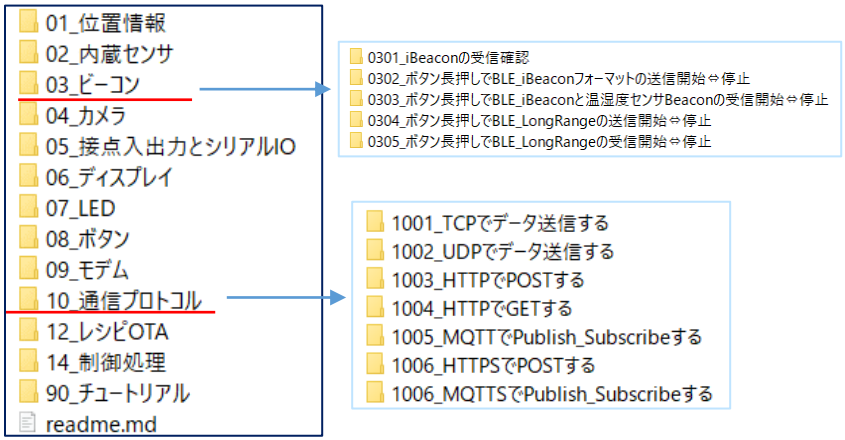
拡張子の種類

拡張子名	説明
*.krs	レシピ言語のソースファイル。
*.kra	sappからAPIを提供するためのヘッダーファイルを作る元ファイル。

サンプルプログラム

SDKに付随するサンプルプログラム

SDK内部の「sample_source」と言うフォルダにプログラムを作成する上で参考となるサンプルプログラムが保存されています。



ここではAPIの基本的な使い方から複数の機能を組み合わせた応用まで、幅広く準備をしていますので、レシピ言語の理解の助けとなるとともに、効率よくプログラムを構築出来ます。

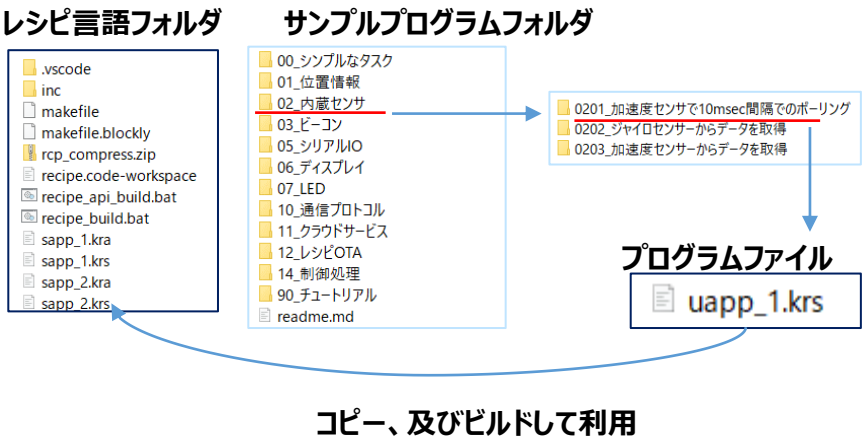
※サンプルプログラムの解説はホームページのサンプルコード紹介の中にもありますので合わせて確認ください。

サンプルプログラムの確認方法

サンプルプログラムの各フォルダの中身を見ると、

の様なファイルが１つ置かれている事が多いです。

このファイルはレシピ言語フォルダの中にそのままコピーして利用できるようにしてありますので、コピー後にビルドして動きを確認することが出来ます。

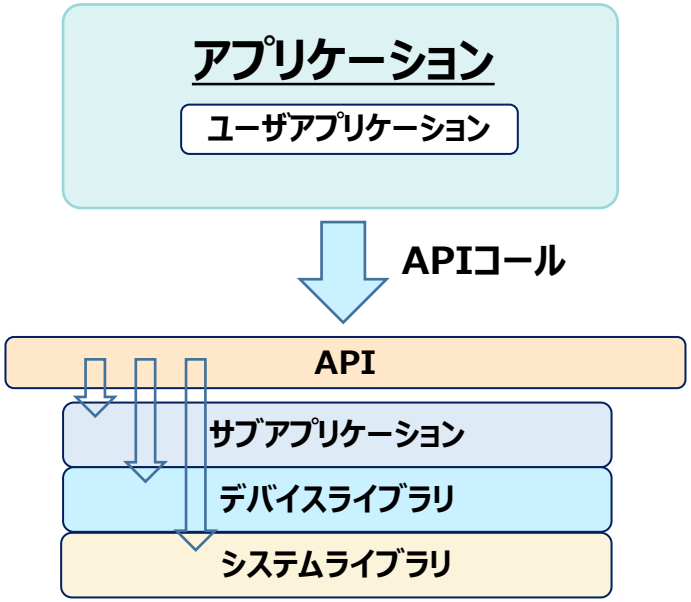


APIについて

レシピ言語には効率良くプログラムを作成するための様々なAPIが準備されています。

APIについて

ドライバやライブラリで提供されている機能は、APIとして準備されており、アプリケーションから利用することで各種制御をすることが出来ます。



APIの種類

様々なAPIが準備されており、以下のような種類があります。

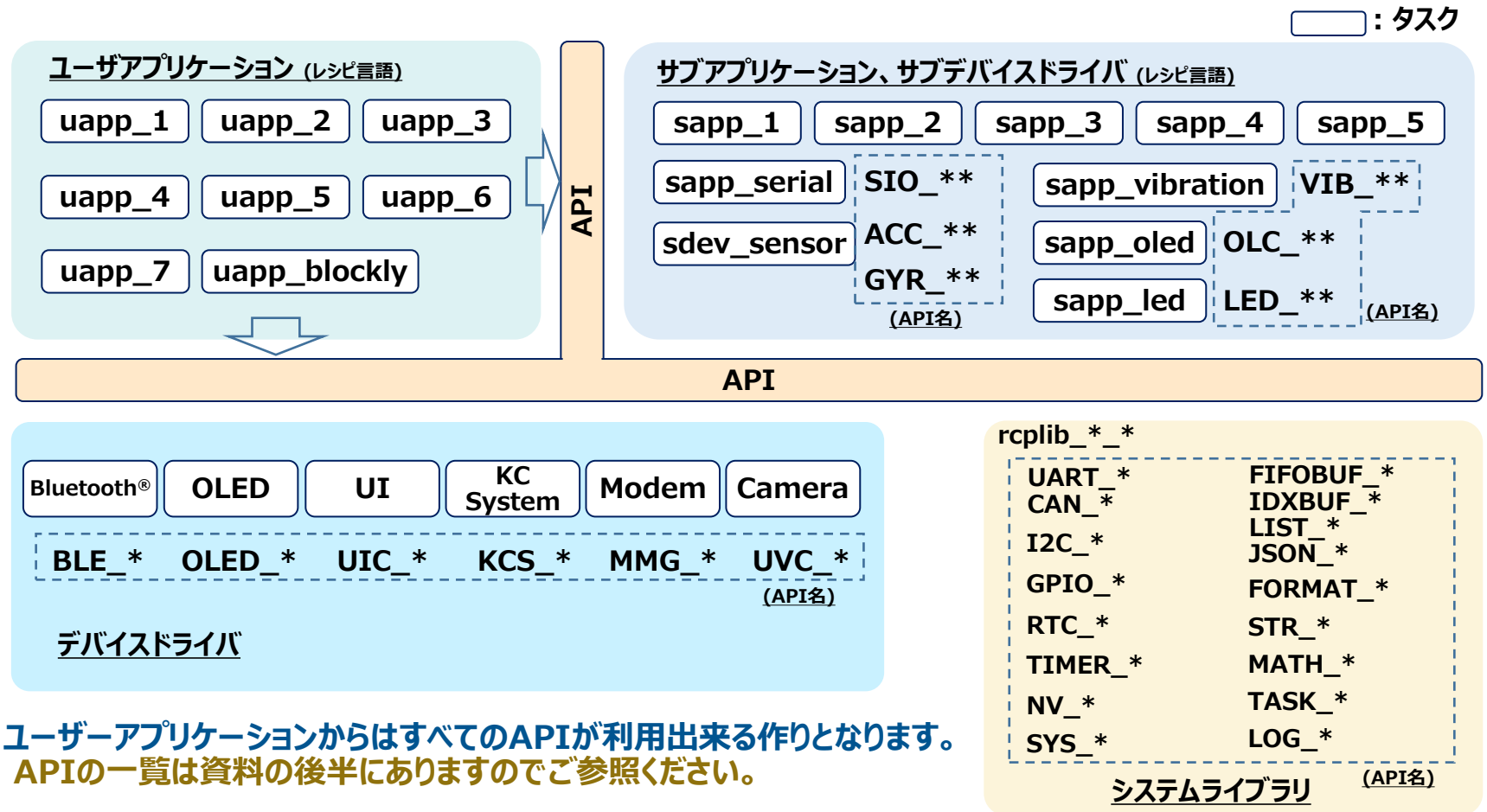
サブアプリケーションAPI
センサー制御
振動検出
OLED制御
LED制御
シリアル制御
Modbus制御

デバイスライブラリAPI
モデム制御
Bluetooth®制御
KCシステム制御
OLED制御
ボタン制御
LED制御
カメラ制御

システムライブラリAPI
システム制御
レシピエンジン情報
タスク制御
不揮発制御
タイマー、RTC制御
GPIO、I2C制御
CAN制御
UART制御
数学演算
文字列操作
フォーマット変換
JSON変数制御
LIST変数操作
FIFOバッファ操作

API構成図

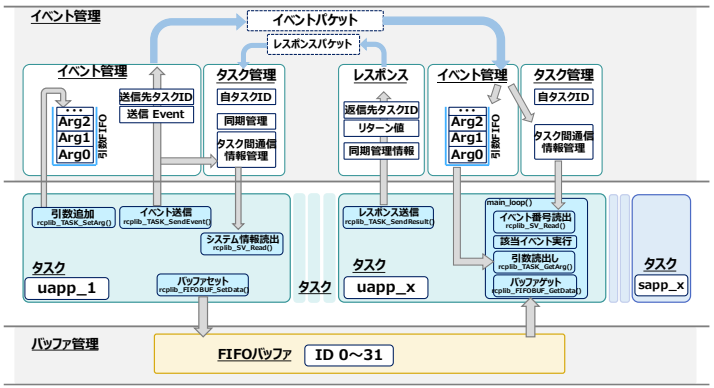
様々な種類のAPIは以下の構成図のように整理されます。



※ユーザーアプリケーションからはすべてのAPIが利用出来る作りとなります。
APIの一覧は資料の後半にありますのでご参照ください。

レシピ言語のタスクとイベント

レシピ言語プログラムの理解にはタスクとタスク間通信の知識が必要となります。
ここではタスク及びタスク間通信について説明します。



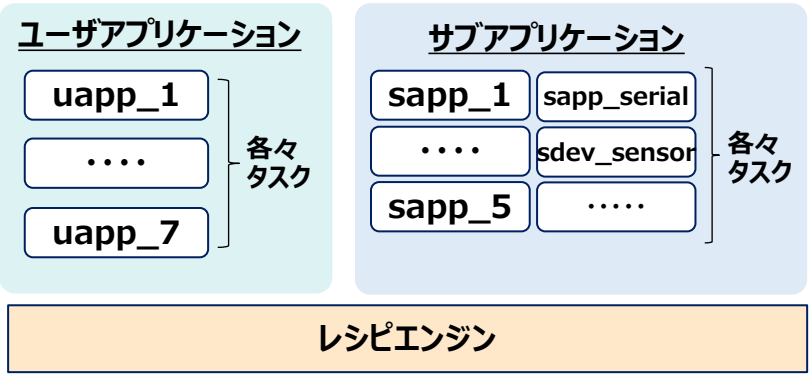
- タスクについて
- タスクの優先度と実行
- イベントの送受信と同期/非同期
- イベント送信のAPI
- FIFOバッファ
- タスク間通信の全体図

タスクについて

レシピ言語のアプリケーション及びサブアプリケーションはタスク単位で動作をします。アプリ間の情報の受け渡しはタスク間通信で行います。

1.タスクについて

レシピ言語のタスクとはReal Time OSのタスクと同義で、並列実行の単位です。また、レシピエンジン上のアプリケーションの単位となります。

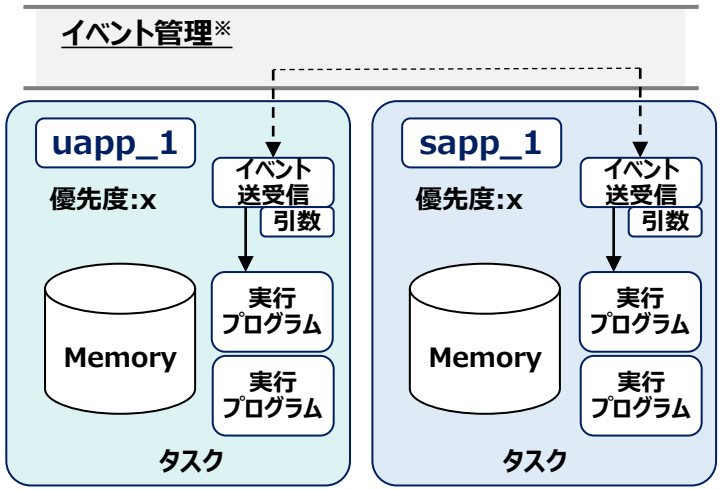


2.タスクの機能

タスクはそれぞれ固有のメモリ空間と優先度を持ち、イベントを受けることで内部のプログラムが実行状態になります。(右図)

2.タスク間通信

タスク間通信にはイベントを用ます。イベントにはイベントに紐づく引数が設定出来、データを送受信することができます。



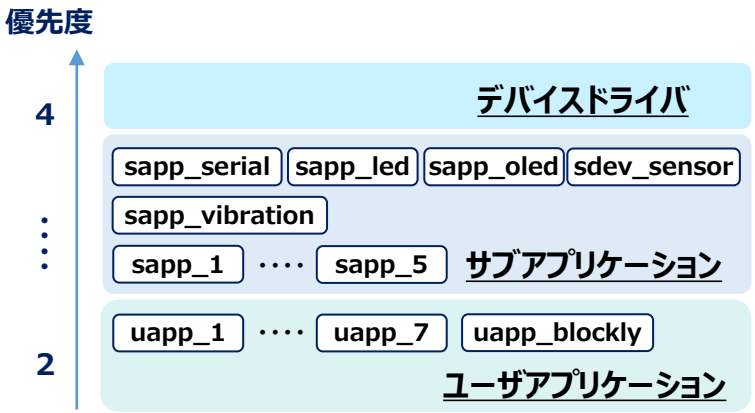
※イベント管理はタスク間でのイベントの管理/調整をするシステム側の制御となります。

タスクの優先度と実行

3.タスクの優先度

タスクは並列処理が行われますが、優先度により処理される順番や頻度が変わります。

優先度、以下のように設計しています。
優先度値が大きいほど優先度が高くなります。

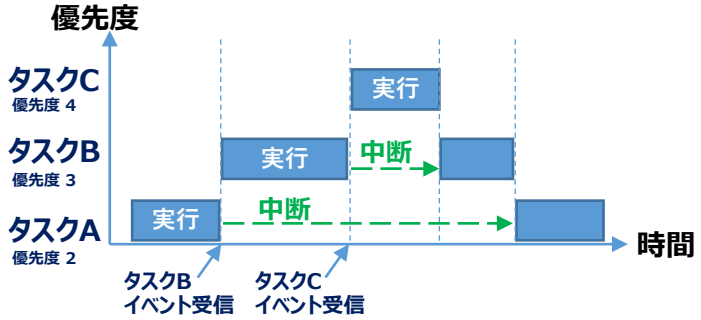


※優先度はタスクのオプション設定で変更が出来ます。
アプリケーションの優先度の変更はタスクの実行順に影響が出ますので、変更時には注意してください。

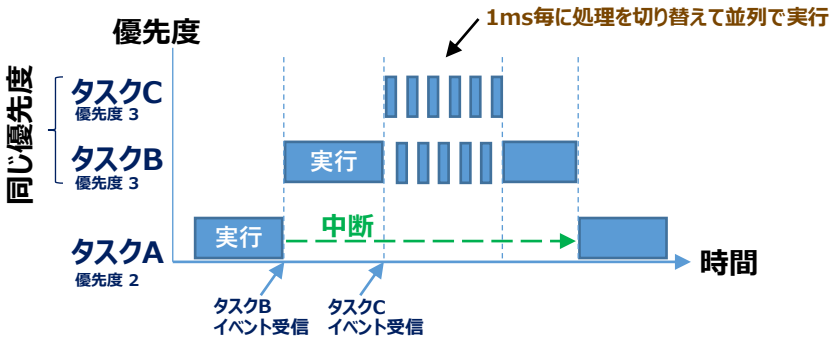
4.タスクの優先度と実行

イベントを受けて実行されるタスクは優先度によって実行の振る舞いは以下のようになります。

優先度の違うタスクのイベント受信時の振る舞い



優先度の同じタスクのイベント受信時の振る舞い

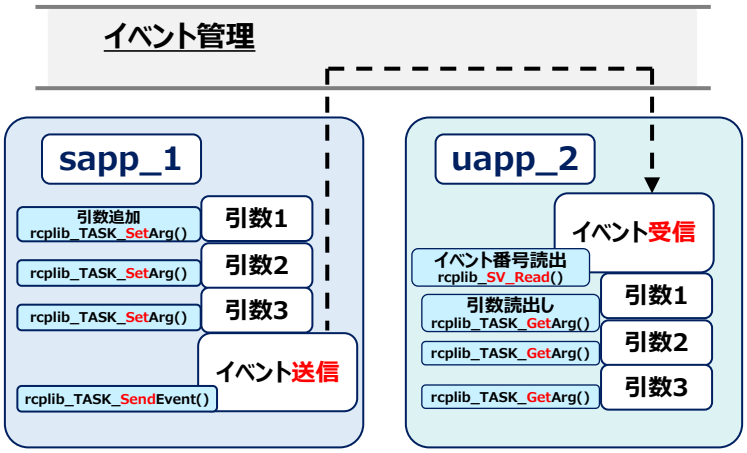


イベントの送受信と同期/非同期

タスク間の連携や情報のやり取りを行う為には、タスク間通信としてイベントを使用します。また、イベントの種類には同期と非同期があります。

5.イベントの送受信

タスク間の連携はイベントの送受信機能を使用します。イベントには送信先タスクの実行トリガを与えることに加え、引数の添付が出来ます。



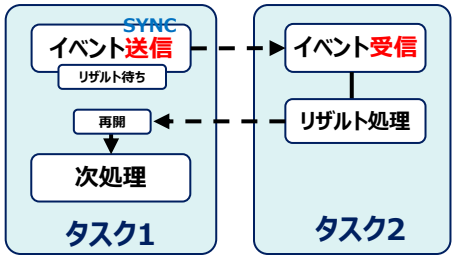
※rcplib_TASK_SetArg()にてセットした引数はrcplib_TASK_GetArg()にて順番に取り出す仕組みになっています。数と順序に気をつけて使用する必要があります。

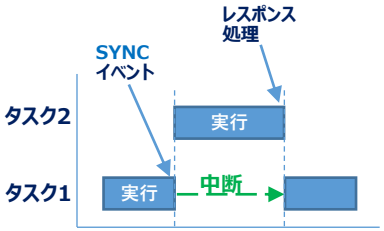
※引数には合計で26byteのサイズしか使えませんので、大きなデータの通知はFIFOバッファ(後述)を利用します。

6.イベントの同期/非同期

イベントを送信する際に相手の応答を待つ同期(SYNC)と応答を待たない非同期(UNSYNC)を選択出来ます。

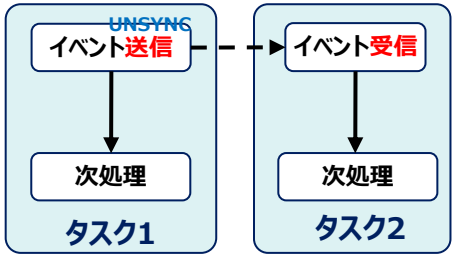
同期(SYNC)の動き

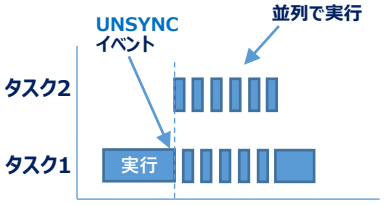




※タスク１と2は同じ優先度の場合

非同期(UNSYNC)の動き





※タスク１と2は同じ優先度の場合

イベント送信のAPI

タスク間でのイベント送受信のAPIは以下となります。

7.イベントの送信API

タスク間でのイベントを送信するためのAPIは
以下のように準備されています。

API

```
rcplib_TASK_SendEvent (
    uint8 task_id,
    uint8 event_id,
    uint8 sync,
    uint8 timeout    );
```

引数

task_id：送り先のタスク番号を指定します。
タスク番号は、taskid.hに定義されている。

event_id：送り先のタスクで処理するイベントの番号を指定します。イベント番号はタスク内ユニークとなるので、設計時に定義をしておきます。

sync：送ったイベントが送信先タスクで処理されるまで次の処理の実行を待つか、待たずに処理を続けるかの選択。(SYNC or UNSYNC)

timeout：syncで送信先タスクで処理待ちをする場合のタイムアウト時間を指定。タイムアウト後はコンテキストが戻り次の処理が実行されます。

タスク間のイベント通知の使用例：

sapp_1

```
//イベントを送信する
rcplib_TASK_SendEvent(
        EAPI_TASKID_UAPP2,        --- 送信先 uapp2
        UAPP2_EVENT_A,            --- uapp2側で定義している番号
        EAPI_UNSYNC_NORESULT,     --- イベント処理待たず
        EAPI_TIMEOUT_ZERO         --- タイムアウトなし
 );
```

uapp_2

```
#define UAPP2_EVENT_A 1          --- 定義はuapp1と共有しておく

//イベントループ
main_loop() {                    --- SV_Readでevent番号取得
evt = rcplib_SV_Read(ENGINE_SVITEM･･･);

if (evt=UAPP2_EVENT_A) {         --- 送られてきたイベントを捕捉
    rcv_event();                 --- 該当のイベント処理を実行
 }
～

//イベント受信関数
func rcv_event() {
   --- 受信処理
}
```

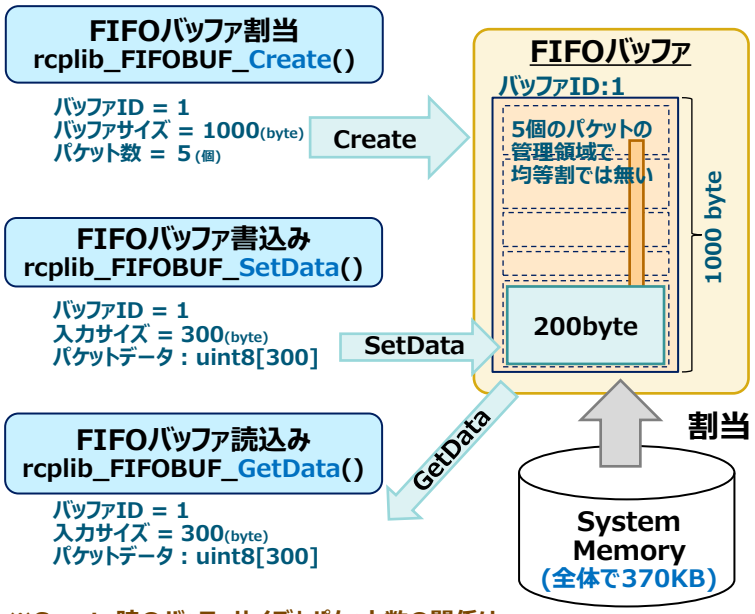
sapp_1で発行されたイベントがuapp_2で受信され、内部関数のrcv_event()が実行される。

FIFOバッファ

イベントで扱える引数のサイズが少ない為、タスク間でデータを受け渡しする場合には、FIFOバッファを使用します。

1.FIFOバッファの仕組み

FIFOバッファはシステムメモリ全体(370KB)の中から割当を行い、システム全体で固有の番号(バッファID)にて使用します。



※Create時のバッファサイズとパケット数の関係は、上記の場合、1000byteを５個のパケットで管理をするという指定となる。(パケットは均等割りでは無い)

2.FIFOバッファのAPI

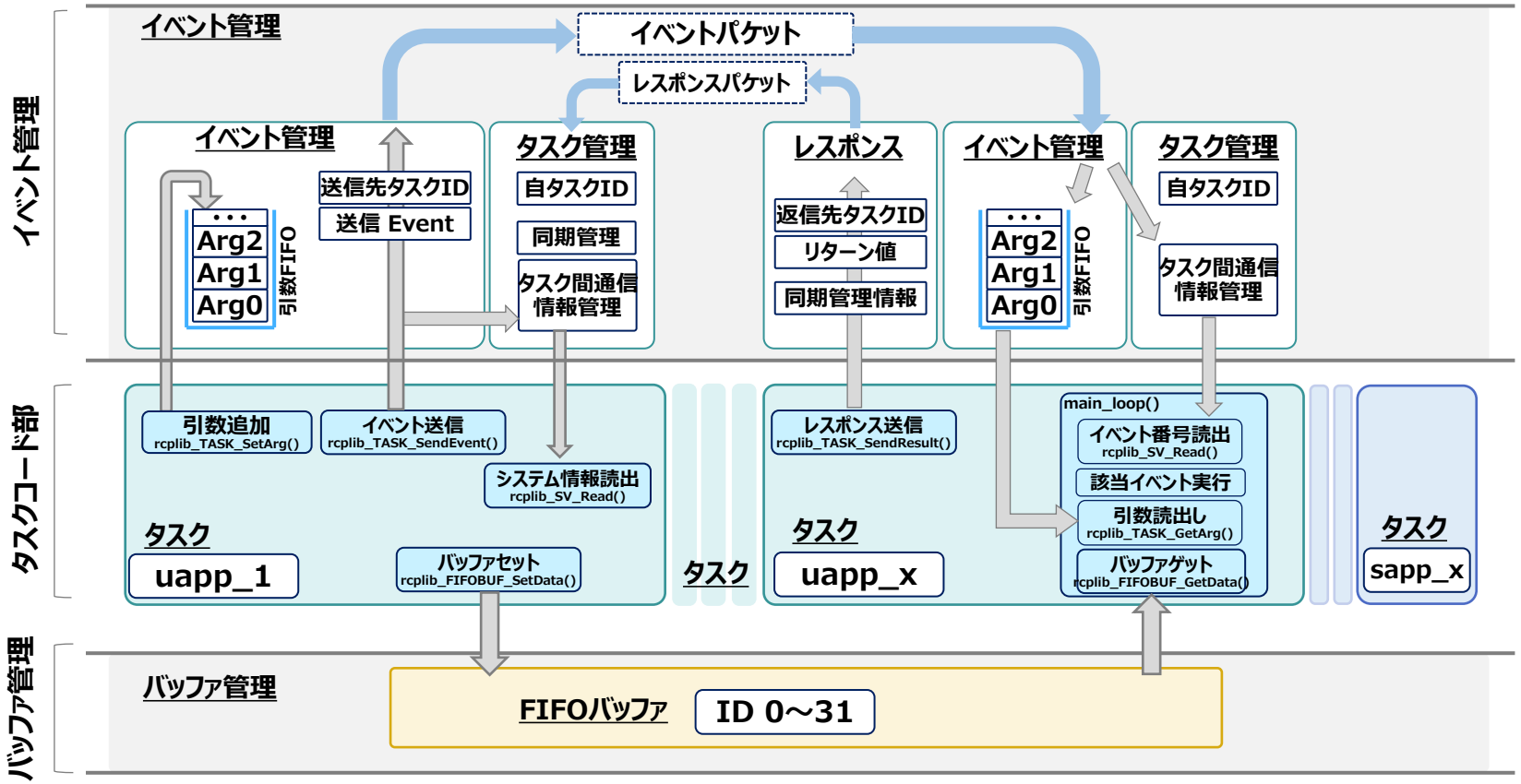
API
```
rcplib_FIFOBUF_Create (
    uint8 fifo_buf_id,
    uint32 buf_size,
    uint16 buf_depth,
    uint8 overwrite    );
```
引数
- fifo_buf_id：FIFOバッファID 0~31 (bufferLib.hで定義)
- buf_size：バッファのトータルサイズ (トータルであることに注意)
- buf_depth：バッファに保存可能なパケット数
- overwrite：バッファフル時の書込み処理の挙動

API
```
rcplib_FIFOBUF_SetData (
    uint8 fifo_buf_id,
    uint16 size,
    uint8[] data);
```
引数
- fifo_buf_id：FIFOバッファID 0~31 (bufferLib.hで定義)
- size：バッファに書き込むサイズ
- data：バッファに書き込むデータの格納配列

API
```
rcplib_FIFOBUF_GetData (
    uint8 fifo_buf_id,
    uint16 size,
    uint8[] data);
```
引数
- fifo_buf_id：FIFOバッファID 0~31 (bufferLib.hで定義)
- size：バッファから読み込むサイズ
- data：バッファから読み込んだデータの格納配列

タスク間通信の全体図

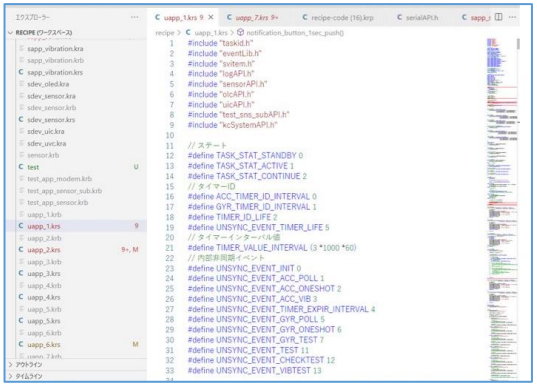
ここまでの内容をふまえて、タスク間通信のイベントとバッファのイメージを下図にまとめます。



※サンプルコード内でのタスク間通信やAPIコール時に多用するので、イメージを捉えておくとソースコードを理解する助けになります。

レシピ言語の言語体系

レシピ言語はC言語に近い言語体系ですが、若干違いがありますので、ここでは言語の基本的な内容を説明します。



- 変数宣言と代入
- データ型の種類
- 定数と記号定数
- 演算子
- 条件分岐命令
- 繰返し命令
- 関数

変数宣言と代入

●数値変数、配列変数の宣言

プログラム中で扱う変数は事前に宣言が必要

書式

```
型名  変数名;          : 数値変数
型名  変数名[添字];    : 配列変数 添字には配列数
```

```
int16 val;              : 16bit 符号付き変数
int32 index[10];        : 32bit 符号付き配列変数
```

●変数への値を代入

変数への代入は直値、演算値、変数が指定可能

書式

```
変数名 = 値;
変数名[添字] = 値;      : または 演算値, 変数
```

```
val = 123;              : 変数に数値を代入
index[0] = 10;          : 配列変数に数値を代入
val = 1 + 2 + 3;        : 演算値を代入
val = index[0];         : 変数を代入
```

注意 変数の宣言と同時の値の代入は不可

✕
```
uint16 val = 123;
uint32 index[5] = {1,2,3,4,5};
```

●文字変数の宣言

文字変数は文字数を添えて宣言

書式

```
str 文字変数名[添字];    : 添字には文字数
```

```
str moji[10];           : 10文字までの文字変数
```

注意 C言語であるchar型は無い

✕
```
char moji[10];
```

●文字変数への文字代入

ダブルクオーテーションで囲った文字を代入可能 又は
str変数への文字列代入は文字列操作APIを利用

書式

```
文字変数名 = “文字列”;
文字変数名 = 文字変数名;
rcplib_STR_SetSting(文字列変数名, “文字列”);
```

```
moji = “abcdef”;                          : 添字の文字数範囲で代入
mcopy = moji;                             : mcopyは文字変数
rcplib_STR_SetSting(moji,“abcdef”);       : mojiはstr型
```

注意 str型は配列変数の様な定義をしますが、配列としての操作は不可。APIを利用。

✕
```
mcopy[1] = moji[1];
mcopy[] = moji[];
```

データ型の種類

●データ型一覧

変数の宣言時に定義できるデータ型は以下の通り

データ型名	説明	値の範囲
int8	符号付き8bit整数	128~127
int16	符号付き16bit整数	32768~32767
int32	符号付き32bit整数	2147483648~2147483647
int64	符号付き64bit整数	9223372036854775808~9223372036854775807
uint8	符号なし8bit整数。	0~255
uint16	符号なし16bit整数	0~65535
uint32	符号なし32bit整数	0~4294967295
uint64	符号なし64bit整数	0~18446744073709551615
float	32bit浮動小数点	1.175494e-38~3.402823e+38
double	64bit浮動小数点	2.225074e-308~1.797693e+308
str	文字列	最大長は65536文字
json	JSON変数	最大サイズは65536byte
list	LIST変数	最大サイズは65536byte

※json, list型の使用方法はマニュアルをご参照下さい。

注意 **ポインタ変数は使えません**

✕ uint8* a;

定数と記号定数

●定数の表記

プログラム中の定数表記は以下の通り

定数表記	使用可能数値	補足	表記例
10進数	0~9の数値	負数は先頭に「-」を指定	123, -123
16進数	0~9,a~f,A~Fの英数字	先頭に「0x」のプリフィクスを指定	0x1a, 0x1A, 0x12ab, 0x12AB
2進数	0~1の数値,「_」区切り文字	先頭に「0b」のプリフィクスを指定	0b10100101, 0b1010_0101
小数	0~9、「.」の数値	指数指定は非対応	3.14
文字	ASCII文字コードのみ	Unicodeは非対応	“abcd”

●記号定数表記

#defineマクロを利用した記号定数が可能

書式 **#define 記号定数名 定数** ：記号定数は英数可能

```
#define PI 3.141592
#define EVENT 0          : 文末のセミコロン無し

cir = diam * PI;
if (ev == EVENT) {       : プログラム中で定数利用
```

●その他定数表記

C言語でつかえる他の定数表記は使用不可

注意 **enum, const, structは使用不可**

✕
```
enum num {1,2,3,4};
const float pi;
```

演算子

●算術演算子

演算子	説明	例
+	加算	a = x + y;
-	減算	a = x – y;
*	乗算	a = x * y;
/	除算	a = x / y;
%	剰余算	a = x % y;

注意 **他の単項演算子,演算代入子の使用不可**

✕ a = x++;　b += x;

●論理演算子

演算子	説明	例
&	AND	a = x & y;
|	OR	a = x | y;
^	XOR	a = x ^ y;
~	反転	a = ~x;

注意 **ビットシフトは使用不可**

✕ a = x << y;　b = x >> y;

●比較演算子

演算子	説明	例
<	小なり	if (a < b)
<=	小なりイコール	if (a <= b)
>	大なり	if (a > b)
>=	大なりイコール	if (a >= b)
==	イコール	if (a == b)
!=	ノットイコール	if (a != b)

●論理演算子

演算子	説明	例
&&	かつ	if ((a==b) && (x==y))
||	または	if ((a==b) || (x==y))

●その他

演算子	説明	説明/例
( )	カッコ	演算優先順位変更 a = (a + b) * x;
[ ]	配列要素指定 アドレス参照	a = x[10]; ：配列の10番目 func_a (x); ：x配列のアドレス渡し

条件分岐命令

●条件分岐命令(if - else)

条件分岐は if ～ else 命令が使用可能

書式

```
if (条件式) {
   条件成立時の処理 ;
}
```

{ は改行後の記載でも可能

数値

```
if ( val == 1 ) {
   val = 0;
}
```

定数との比較の条件式

```
if ( moji == “abcd” ) {
   moji = “cdef”;
}
```

文字

変数との比較の条件式

```
if ( val1 == val2 ) {
   val = 0;
}
```

```
if ( moji1 == moji2 ) {
   moji1 = “cdef”;
}
```

※str変数同士の比較可能

注意 **switch – case文は使用不可**
if – else文で記述必要

✕

```
switch (式) {
   case1 定数1 :
     処理;
     break;
   case2 定数2 :
     :
```

⇨

```
if (条件式) {
     処理;
} else if (条件式) {
     処理;
} else if (条件式) {
     :
```

書式

```
if (条件式) {
   条件成立時の処理 ;
} else {
   条件不成立時の処理 ;
}
```

```
if (条件式1) {
   条件1成立時の処理 ;
} else if (条件式2) {
   条件2成立時の処理 ;
} else {
   条件1,2不成立時の処理 ;
}
```

```
if ( age < 10 ) {
   moji = “under 10”;
} else {
   moji = “other”;
}
```

```
if ( temp < 15 ) {
   moji = “cold”;
} else if ( temp < 25) {
   moji = “warm”;
} else {
   moji = “hot”;
}
```

注意 **カッコ { } は省略不可**

✕

```
if (条件式)
   処理;
else
   処理;
```

(C言語では処理が1行の場合省略可)

繰返し命令

●繰返し命令 (while)

繰返し命令は while命令が使用可能

書式

```
while (条件式) {
   条件成立時の処理 ;
}
```

条件式は「継続条件」

{ は改行後の記述でも可能

```
i = 0;
while ( i < 10 ) {
   処理;
   i = i + 1;
}
```

: i が0-9までの間ループ
終了条件を満たすように
ループの中でケアをしないと
無限ループになるので注意

```
while (i < 10) {
  while (k < 20) {
    …
```

: ループの入れ子も可能

注意 **for文, do..while文は使用不可**

✕
```
for (i=0; i<10; i=i+1)
{
  処理;
}
```

✕
```
do {
  処理;
  i=i+1;
} while (i<10);
```

break, continue 文も使用可能

書式

```
while (条件式) {
   if (条件) {
      処理1 ;
      break;
   }
   処理2 ;
}
```

ループを終了

```
while (条件式) {
   if (条件) {
      処理1 ;
      continue;
   }
   処理2 ;
}
```

ループの途中の
処理をスキップ

```
i = 0;
while ( i < 10 ) {
   if (age[i]>10) {
      処理;
      break;
   }
   i = i + 1;
}
```

```
i = 0;
while ( i < 10 ) {
   if (age[i]>10) {
      i = i + 1;
      処理1;
      continue;
   }
   処理2;
   i = i + 1;
}
```

continue時はカウンターも飛ばすので注意が必要

関数

●関数 (func)

関数は func キーワードで定義可能

書式

```
func 関数名(データ型 引数名, …)
{
   関数内処理;
   return(戻り値);
}
```

引数は複数指定可能
省略も可能

returnは()が必要
(戻り値はint32型)

```
func doSome ()
{
   関数内処理;
   return(0);
}
```

引数がない場合

```
func TriangleArea (
   uint16 base,
   uint16 height )
{
   uint16 area;
   area = base * height / 2;
   return(area);
}
```

引数は関数内で利用可能

ローカル変数

※引数名, ローカル変数名のスコープは関数内

関数の呼び方

```
ans = TriangleArea (teihen, 10);
```

returnの返り値が入る

引数は 変数,直値 可能

```
TriangleArea (teihen, 10);
```

返り値は破棄される

配列変数、文字変数の引数

書式

引数で渡されたの数値配列変数はアドレスで通知がされ、文字変数は渡された先でコピーされる。

```
func 関数名(
  データ型 配列引数名[])
{
   関数内処理;
   return(戻り値);
}
```

```
func 関数名(
  str 変数名[添字])
{
   関数内処理;
   return(戻り値);
}
```

※数値配列と文字列でメモリの扱いが違うので注意

数値

```
func myf(uint8 ap[])
{
  ap[0] = 3;
  ap[1] = 4;
  …
}
```

apという名前で呼び元と同じメモリを使用

呼び元

```
uint8 val[5];
val[0]=1; val[1]=2;
myf(val);
```

アドレスとして渡す
関数戻った後はap[0]は3となる。

文字

```
func myf(str mp[10])
{
  mp = “cdf”;
  …
}
```

添字が必要

mpという新しいメモリを確保してコピーされる。

呼び元

```
str moji[5];
moji = “abc”;
myf(moji);
```

文字列として渡す
関数戻った後はmojiは”abc”のまま。

C言語との比較

機能		C言語	レシピ言語
定数	整数	◎	◎
	小数	◎	◎
	文字	◎	◎
変数	int8/uint8	◎	◎
	int16/uint16	◎	◎
	int32/uint32	◎	◎
	int64/uint64	◎	◎
	float	◎	◎
	double	◎	◎
	文字	○*1	◎
	文字最大長	制限なし	65535
	名称長	制限なし	制限なし
	定義数	制限なし	65535
算術演算	四則剰余	◎	◎
	括弧	◎	◎
	冪乗	◎	×
	冪根	◎	◎
	関数呼び出し	◎	◎
論理演算	and	◎	◎
	or	◎	◎
	xor	◎	◎
	not	◎	◎
	shift	◎	×*2
	比較(=,<,>)	◎	◎
	論理積(AND)	◎	◎
	論理和(OR)	◎	◎
	否定(NOT)	◎	◎

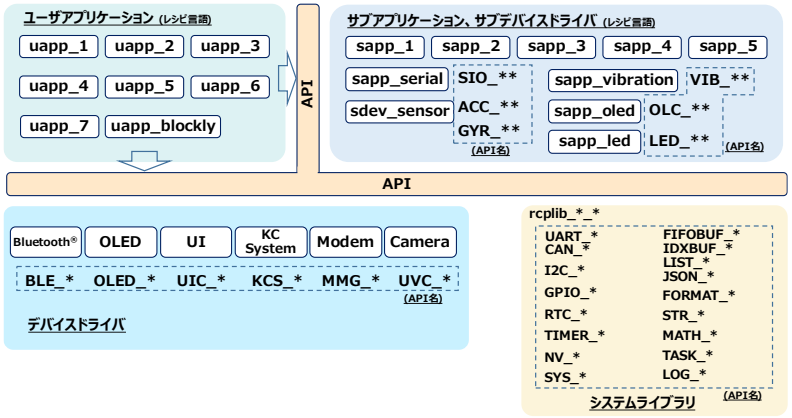
*1 char[]を文字として利用する
*2符号なし整数の乗除演算にて代用可

機能		C言語	レシピ言語
配列定義	整数	◎	◎
	実数	◎	◎
	文字	◎	○*3
データ定義	struct	◎	×
	union	◎	×
	enum	◎	×
	#define	◎	◎
	#ifdef/#else/#endif	◎	◎
	ポインタ	◎	○*4
	json	×	◎
データ制御	if/else/else if	◎	◎
	while/for	◎	○*5
	break/continue	◎	◎
	sleep	◎	◎
	ログ出力	◎	◎
	文字列操作	◎	◎
	フォーマット変換(atoi/sprintf)	◎	◎
	FIFOバッファ操作	×*6	◎
	リングバッファ操作	×*6	◎
関数	関数定義	◎	◎
	関数内変数(ローカル変数)	◎	◎
	引数渡し	◎	◎
	戻り値渡し	◎	◎

*3 文字の配列定義はLIST変数にて代用可
*4 配列変数のみアドレス参照可
*5 whileのみ利用可
*6 外部ライブラリを利用することが前提

APIの構成と各種API

レシピ言語にはメモリ操作や数学関数、タイマなどのシステム関連のAPIや、
Modem,シリアルなどデバイスを制御するAPIが豊富に準備されています。
ここではこれらAPIの構成とAPIの種類について説明します。



- APIの構成
- システムAPI
- デバイスAPI
- サブアプリAPI

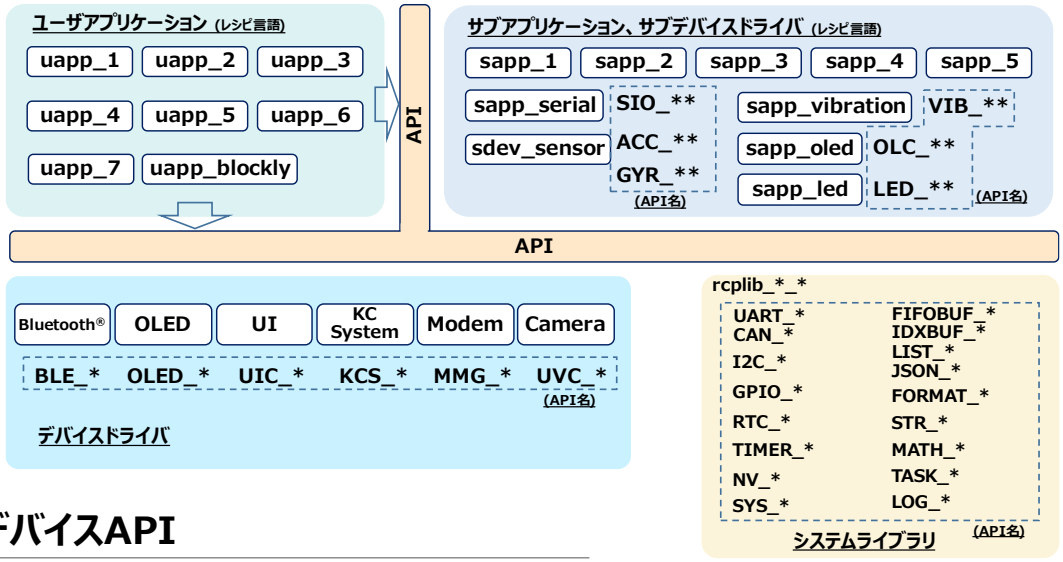
APIの構成

APIは各機能部から提供され、ユーザーアプリケーション(uapp)からはすべてのAPIが利用可能な作りになっています。(※一部非公開あり)

**ユーザーアプリケーションからは
すべてのAPIを利用することが可能**

サブアプリAPI

一部のアルゴリズムやプロトコルはレシピ言語で変更が出来るようサブアプリ/サブドライバで組まれて利用しやすい様にAPI提供されています。
(自身でAPIを作成することも可能です。)



デバイスAPI

ハードウェアに近い部分、複雑な処理はネイティブコードで組まれ、各機能毎のAPIとして提供されています。

システムAPI

文字処理やバッファ処理、直接ハードウェアを制御する部分はシステムAPIとして提供されます。

システムAPI

タイマー処理や文字処理、バッファ処理などシステムとして提供される機能をAPIとして定義しています。ハードウェアを直接制御するAPIも含まれます。

●システム制御

API	説明
rcplib_SYS_Reboot	端末再起動
rcplib_SYS_VersionENGINE	レシピエンジンバージョン情報取得
rcplib_SYS_VersionSTM	機器のFwバージョン情報取得
rcplib_SYS_GetTick	システムチックを取得
rcplib_SYS_VendorCode	ベンダーコード取得
rcplib_SYS_Sleep	指定時間スリープ
rcplib_SYS_LogPrint	ログ出力
rcplib_LOG_Print	USBデバッグログ出力
rcplib_LOG_Low	USBデバッグログ出力(Lowレベル)
rcplib_LOG_Blockly	ログ出力(Middleレベル - Flash出力)
rcplib_LOG_Flash	ログ出力(Highレベル - Flash出力)
rcplib_SYS_StoV	文字列変数を数値に変換
rcplib_ALPM_IsEnterLPM	LowPowerModeへの移行か判断
rcplib_PWRM_SetPerformanceMode	CPUモード変更
rcplib_KCS_RcpOTA	レシピのOver The Air機能を実行

●レシピエンジン情報

API	説明
rcplib_SV_Read	タスク間通信関連情報読出し

●タスク制御

API	説明
rcplib_TASK_InitArg	タスク送受信時の引数管理情報を初期化
rcplib_TASK_GetArg	相手タスクから渡された引数情報を取得
rcplib_TASK_SetArg	データとその型情報を引数情報に追加
rcplib_TASK_SendEvent	指定したタスクIDにイベントを送信
rcplib_TASK_SendResult	同期イベントを受信した後の返答を送信

●不揮発制御

API	説明
rcplib_NV_Read	不揮発アイテムのデータを読み出す
rcplib_NV_Write	ミラー領域の不揮発アイテムに書き込む
rcplib_NV_Sync	ミラー領域のデータを不揮発メモリに反映

システムAPI

●タイマー,RTC制御

API	説明
rcplib_TIMER_Create	ソフトタイマーを作成する
rcplib_TIMER_Start	ソフトタイマーをスタートする
rcplib_TIMER_Stop	ソフトタイマーをストップする
rcplib_TIMER_SetPeriod	ソフトタイマーの周期を変更
rcplib_TIMER_IsActive	ソフトタイマーの動作状態を取得
rcplib_RTC_GetTime	RTC時刻をUNIX時間(UTC)で取得
rcplib_RTC_CreateAlarm	RTCアラームを作成＆設定
rcplib_RTC_DeleteAlarm	RTCアラームを削除
rcplib_RTC_ChangeAlarm	RTCアラーム時刻を再設定

●CAN制御

API	説明
rcplib_CAN_Init	CANポートを初期化
rcplib_CAN_DeInit	CANポートを解除
rcplib_CAN_ConfigGlobalFilter	CANのグローバルフィルタを設定
rcplib_CAN_ConfigFilter	CANの個別フィルタを設定
rcplib_CAN_Start	CANを開始
rcplib_CAN_Stop	CANを停止
rcplib_CAN_ActivateNotification	CANの受信通知を有効化
rcplib_CAN_DeactivateNotification	CANの受信通知を無効化
rcplib_CAN_Transmit	CANのデータ送信
rcplib_CAN_AbortTxRequest	CANのデータ送信を中断

●GPIO

API	説明
rcplib_GPIO_Read	GPIO Read
rcplib_GPIO_Write	GPIO Write

●UART(RS-485,RS-232C)制御

API	説明
rcplib_UART_Init	UARTポート初期化
rcplib_UART_DeInit	UARTポート解除
rcplib_UART_SetTxFifoThreshold	UART送信FIFO閾値設定
rcplib_UART_SetRxFifoThreshold	UART受信FIFO閾値設定
rcplib_UART_SetFifoMode	UARTのFIFOモードの有効/無効を設定
rcplib_UART_Transmit	UART送信

システムAPI

●数学演算

API	説明
rcplib_MATH_Sin	正弦(sin)関数
rcplib_MATH_Cos	余弦(cos)関数
rcplib_MATH_Tan	正接(tan)関数
rcplib_MATH_Asin	逆正弦(arcsin)関数
rcplib_MATH_Acos	逆余弦(arccos)関数
rcplib_MATH_Atan	逆正接(arctan)関数
rcplib_MATH_Log	自然対数(log)関数
rcplib_MATH_Exp	指数(exp)関数
rcplib_MATH_Log10	常用対数(log10)関数
rcplib_MATH_Sqrt	平方根(sqrt)関数
rcplib_MATH_Pow	べき乗(pow)関数
rcplib_MATH_Abs	絶対値(abs)関数

●文字列操作

API	説明
rcplib_STR_Cat	2つの文字列変数を連結
rcplib_STR_Len	文字列変数の文字数を取得
rcplib_STR_Find	文字列変数内のキーワード有無を検索
rcplib_STR_Cut	文字列変数の指定位置に指定長の文字列を書き込む
rcplib_STR_Replace	文字列内の文字を別の文字に置き換える
rcplib_STR_SetString	文字列変数に任意の文字列を書き込む

●フォーマット変換

API	説明
rcplib_FORMAT_Time	UTC時刻を時刻関連の文字列変数に変換
rcplib_FORMAT_Binary	任意のデータをuint8配列に変換
rcplib_FORMAT_Value	uint8配列のデータを任意の型に変換
rcplib_FORMAT_String	任意のデータをstr配列に変換
rcplib_FORMAT_EncPercent	文字列をパーセント表記の文字列変数に変換
rcplib_FORMAT_EncWWWForm	文字列をx-www-form-urlencodedの文字列変数に変換
rcplib_FORMAT_EncBase64	Base64にエンコード
rcplib_FORMAT_DecBase64	Base64をデコード

●JSON変数制御

API	説明
rcplib_JSON_SearchK	jsonから指定したkeyの存在を確認
rcplib_JSON_SearchN	json内の指定したkeyのvalueを数値形式で取得
rcplib_JSON_SearchS	json内の指定したkeyのvalueを文字列形式で取得
rcplib_JSON_UpdateV	json内の指定したkeyのvalueのデータを書き換え
rcplib_JSON_UpdateK	json内のkeyのvalueのデータを文字列で書き換え
rcplib_JSON_Clear	json変数をクリアする
rcplib_JSON_Delete	json変数内の指定したkeyを消去
rcplib_JSON_IsEmpty	json変数のデータの有無を確認
rcplib_JSON_GetEntry	json変数のデータ数を取得

システムAPI

●LIST変数操作

API	説明
rcplib_LIST_Clear	list変数をクリア
rcplib_LIST_SearchN	list変数内の指定したindexのvalueを数値形式で取得
rcplib_LIST_SearchS	list変数内の指定したindexのvalueを文字列形式で取得
rcplib_LIST_Update	list変数内の指定したindexのvalueのデータを書き換え
rcplib_LIST_Delete	list変数内の指定したindexデータを消去
rcplib_LIST_IsEmpty	list変数のデータの有無を確認
rcplib_LIST_GetEntry	list変数のデータ数を取得
rcplib_LIST_GetRemainSize	list変数の残りのバッファサイズを取得

●FIFOバッファ操作

API	説明
rcplib_FIFOBUF_Create	FIFOバッファを定義
rcplib_FIFOBUF_SetData	FIFOバッファにデータパケットを書き込む
rcplib_FIFOBUF_SetFormatData	FIFOバッファにFormat定義したデータパケットを書き込む
rcplib_FIFOBUF_GetData	FIFOバッファのデータパケットを読み出す
rcplib_FIFOBUF_PeekData	FIFOバッファのデータパケットを読み出す
rcplib_FIFOBUF_DefineFormatData	SetFormatData()で利用するFormatを定義
rcplib_FIFOBUF_GetEntryNumber	FIFOバッファ内のデータパケット数を取得

●インデックスバッファ操作

API	説明
rcplib_FIXBUF_Pop	Fixedバッファのデータを読み出す
rcplib_IDXBUF_Free	Indexバッファを解放
rcplib_IDXBUF_Push	Indexバッファの全データをクリア
rcplib_IDXBUF_Pop	Indexバッファのデータを読み出す
rcplib_IDXBUF_Peek	Indexバッファの指定したデータを読み出す
rcplib_IDXBUF_Update	Indexバッファの指定したデータを更新
rcplib_IDXBUF_Delete	Indexバッファの指定したデータを削除
rcplib_IDXBUF_GetEntryNumber	Indexバッファ内のデータ数を取得
rcplib_IDXBUF_GetEntryData	Indexバッファ内のデータを文字列に出力
rcplib_IDXBUF_GetRemainSize	Indexバッファの未利用のサイズを取得
rcplib_IDXBUF_IsEmpty	Indexバッファ内のデータ有無を取得
rcplib_IDXBUF_Clear	Indexバッファの格納データを消去

デバイスAPI

ハードウェアに近い部分や複雑な処理はデバイスAPIとして、C言語で組まれ、各機能毎に提供されています

●モデム制御

API	説明
MMG_SetDelayOfftime	利用が無い場合のModemを電源OFFするまでの時間を設定
MMG_GetTime	基地局からのUNIX時間を取得
MMG_GetRevision	ModemのFirmwareバージョンを取得
MMG_GetAntLevel	Modemのアンテナピクトレベルを取得
MMG_GetIMEI	ModemからIMEIを取得
MMG_GetIMSI	ModemからIMSIを取得
MMG_GetNUM	Modemから電番を取得
MMG_SetConnectParam	ソケット接続の基本設定を行う
MMG_SetReceiveParam	ソケット切断時のイベント受付の基本設定を行う
MMG_ConnectSOCKET	ソケット接続/切断を行う
MMG_SendSOCKET	「UDP/TCP/TLS」でデータ送信を行う
MMG_SetConnectParamHTTP	HTTP(S)の基本設定を行う
MMG_SendHTTP	HTTP(S)でデータ送信を行う
MMG_SetConnectParamMQTT	MQTT(S)の接続基本設定を行う
MMG_SetSubscriveParamMQTT	MQTT(S)のSubscribe基本設定を行う
MMG_ConnectMQTT	MQTT(S)接続/切断を行う
MMG_SendMQTT	MQTT(S)でデータ送信(Publish送信)を行う
MMG_GetGNSS	GNSS位置情報を取得する
MMG_SetCertificate	TLS認証情報(CA証明書/クライアント証明書/秘密鍵/PSK)を設定

●Bluetooth®制御

API	説明
BLE_SettingDataSet	送受信機能の基本設定を行う
BLE_FilterDataSetBeacon	iBeacon受信機能のフィルタ設定を行う
BLE_FilterDataSetSensor	SensorBeacon受信機能のフィルタ設定を行う
BLE_FilterDataSetLongRange	LongRangeのBeacon受信機能のフィルタ設定を行う
BLE_StartPolling	iBeaconとSensorBeaconの受信を開始する
BLE_StartPollingLongRange	LongRangeの受信を開始する
BLE_StartOneShot	iBeaconとSensorBeaconの1shot受信を開始する
BLE_StartOneShotLongRange	LongRangeの1shot受信を開始する
BLE_StartPDUSending	Advertise送信を開始する
BLE_ScanStop	送受信要求を解除する
BLE_GetRevision	モジュールのFirmwareバージョン情報を取得する

デバイスAPI

●システム制御

API	説明
KCS_Shutdown	端末の電源をOFFにする要求を行う

●Pushボタン

API	説明
UIC_SenseButtonS	プッシュボタン短押し(0.1～1秒押し)時のリスナー登録
UIC_SenseButtonM	プッシュボタン長押し(1～4秒押し)時のリスナー登録
UIC_SenseButtonL	プッシュボタン長押し(4～10秒押し)時のリスナー登録

●OLED制御

API	説明
OLED_Power	OLEDの電源を制御する
OLED_DisplayTimeNotationChange	「動作の状況」を確認する画面の時刻表示を切り替える

●カメラ制御

API	説明
UVC_RegistUsbVideoStatusListener	USB制御ステータス変化通知のリスナー登録
UVC_BufCtrlSmpRate	USBカメラの撮影画像を動画バッファに記録する周期を設定
UVC_VideoStreamCtrl	USBカメラを制御

サブアプリAPI

一部のアルゴリズムやプロトコルなどはレシピ言語で変更が出来るようにサブアプリ/サブドライバで組まれており、利用しやすい様にAPI提供されています

●センサー制御

API	説明
SNS_SetDelayOffTime	指定した秒数経過後にセンサの電源をOFFする
ACC_GetDataOneShot	加速度センサからデータを取得する
ACC_SetDeviceMeasure	加速度センサのパフォーマンスモードを変更する
ACC_SetVibrationMode	振動検出の有効化無効化を設定
ACC_StartPolling	加速度センサのポーリングを実施する
GYR_GetDataOneshot	ジャイロセンサからデータを取得する
GYR_StartPolling	ジャイロセンサのポーリングを実施する
ACC_SetDeviceRange	加速度センサのダイナミックレンジ変更
GYR_SetDeviceRange	ジャイロセンサのダイナミックレンジ変更

●振動検出

API	説明
VIB_SetSenseMode	振動検出を設定する

●OLED制御

API	説明
OLC_DisplayPict	OLEDにシステム画面を表示する
OLC_DisplayChar	OLEDに文字を表示する
OLC_DisplayFree	OLEDに任意のピクセルを表示する

●LED制御

API	説明
LED_DoLightup	LEDを点灯させる
LED_DoBlink	LEDを点滅させる

サブアプリAPI

●シリアル制御

API	説明
SIO_Regist	データ受信時のコールバックイベント番号を登録
SIO_SetSeparateModeCode	データ受信時の改行文字を登録
SIO_SetSeparateModeSize	データ受信時のデータサイズを登録
SIO_SetUartMode	UART通信設定
SIO_SetCanMode	CAN通信設定
SIO_SendDataSerial	RS485/RS485[2WIRE]/RS232Cデータ送信
SIO_SendDataCAN	CANデータ送信

●Modbus制御

API	説明
SIO_ModbusReadCoilStatus	コイルステータス読出し
SIO_ModbusReadInputStatus	インプットステータス読出し
SIO_ModbusReadHoldingRegister	保持レジスタ読出し
SIO_ModbusReadInputRegister	入力レジスタ読出し
SIO_ModbusForceSingleCoil	コイル情報の書換え
SIO_ModbusWriteSingleRegister	保持レジスタの書換え
SIO_ModbusDiagnostics	診断コマンド
SIO_ModbusWriteMultipleRegisters	複数の保持レジスタの書換え

THE NEW VALUE FRONTIER

KYOCERA

京セラ株式会社

※本サンプルコードは動作確認を行なっておりますが、全ての環境において動作を保証するものではありません。正しく動作することを確認の上でご利用ください。

※ 本資料は2022年10月現在のものです。
※ LTEは、ETSIの商標です。
※ PowerShell, Windowsはマイクロソフト グループの企業の商標です。
※ Linux®は、米国およびその他の国におけるLinus Torvaldsの登録商標です。