コードとモデル図を対応づけてみよう

ETロボコン実行委員会 – er-info@etrobo.jp – 2.0, 2016-05-21 12:56:41 | 2016年用

sample00（ウォーカー）を動かしてみよう

sample00をビルドして動かしてみましょう。

どんな動きをするプログラムでしょうか。

sample00/app.cpp

```
#include "app.h"
#include "util.h"

#include "Motor.h"
#include "Clock.h"

using namespace ev3api;

/**
 * メインタスク
 */
// tag::main_task_1[]
void main_task(intptr_t unused) {

  Motor leftWheel(PORT_C);
  Motor rightWheel(PORT_B);
  Clock clock;

  const int8_t pwm = (Motor::PWM_MAX) / 6;
  const uint32_t duration = 2000;

  init_f(__FILE__);
  while(1) {
    msg_f("Forwarding...", 1);
    leftWheel.setPWM(pwm);
    rightWheel.setPWM(pwm);
    clock.sleep(duration);
// end::main_task_1[]
// tag::main_task_2[]
    msg_f("Backwarding...", 1);
    leftWheel.setPWM(-pwm);
    rightWheel.setPWM(-pwm);
    clock.sleep(duration);

    // 左ボタンを長押し、それを捕捉する
    if (ev3_button_is_pressed(LEFT_BUTTON)) {
      break;
    }
  }

  msg_f("Stopped.", 1);
```

```
42    leftWheel.stop();
43    rightWheel.stop();
44    while(ev3_button_is_pressed(LEFT_BUTTON)) {
45      ;
46    }
47
48    ext_tsk();
49  }
50  // end::main_task_2[]
```

どんな動作をするプログラム？

- 2000msごとに、前進と後退を繰り返す
- 左ボタンが押されたのを捕捉すると停止する

それはプログラムのどこに書いてありますか

- 前進する、後退するという処理はどれですか
- ボタンが押されるという処理はどれですか

その前に…

そもそも、何をしているのかプログラムから読み取れますか？

図に表して確認してみましょう。

サンプルモデル図のプロジェクトを作る

astah*でモデル図を作成するプロジェクトを用意します。

1. デスクトップに演習用のディレクトリ（フォルダ）「beginners」を作ります
2. astah* Professionalを起動します（ライセンスは設定できていますか）
3. 「ファイル」→「プロジェクトの新規作成」でプロジェクトを作成します
4. プロジェクトのファイル名は「sample00」にしましょう
5. 「ファイル」→「プロジェクトを保存」で、「beginners」ディレクトリに保存します

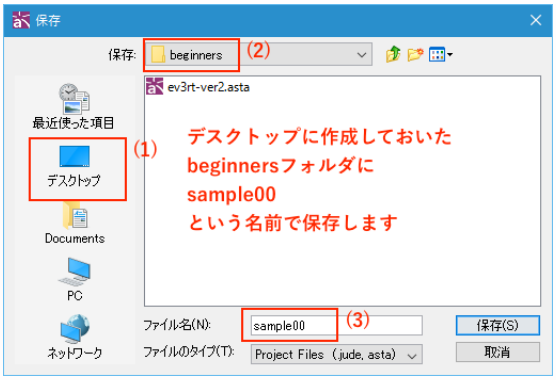

図 1. プロジェクトを保存する

ライブラリの参照プロジェクトを追加する

EV3RTのC++用のクラス群が定義されているプロジェクトを用意してあるので、これを自分が作成したプロジェクトから参照できるようにします。

1. 「ファイル」→「参照プロジェクト管理」で参照プロジェクト管理ダイアログを開きます
2. 「追加」ボタンで「ファイルの指定」ダイアログを開きます
3. 「相対パス」をチエックして、ファイル名に「ev3rt-verX.asta」を指定し「了解」ボタンを押します
4. 参照プロジェクト管理ダイアログを閉じます
5. 構造ツリーに「ev3api」というパッケージが追加されていることを確認しましょう

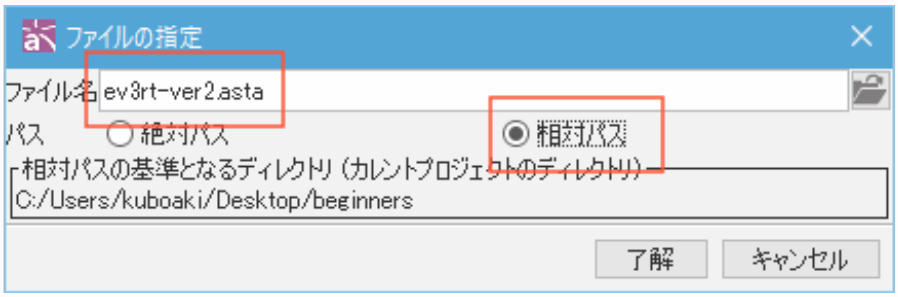

図 2. 参照プロジェクトとして「ev3rt-verX.asta」を相対指定する

コードの登場人物をオブジェクト図に表す

sample00のコードを、次の手順でそのまま図に表してみましょう。

1. 「図」→「クラス図」でクラス図を追加し、図の名前を「sample00のオブジェクト図」にします
2. クラス図のパレットから「インスタンス仕様」を選択し、アイコンが反転したら、マウスをダイアグラムエディタ上でクリックします
3. 「インスタンス仕様0」を、sample00.cppで使っているインスタンス「leftWheel」に変更します

4. 同様にして、コード上にある「rightWheel」「clock」も作成します
5. 同様にして「main_task」も作成します
    - main_taskはRTOSから呼び出されるCの関数ですが、この演習では呼び出す側のクラスのインスタンスとみなして扱うことにします
6. 呼び出している側と呼びだされている側の間にリンクを引きます

作成したsample00のオブジェクト図１

この図では各インスタンスの「スロットの表示」を「オフ」にしています。

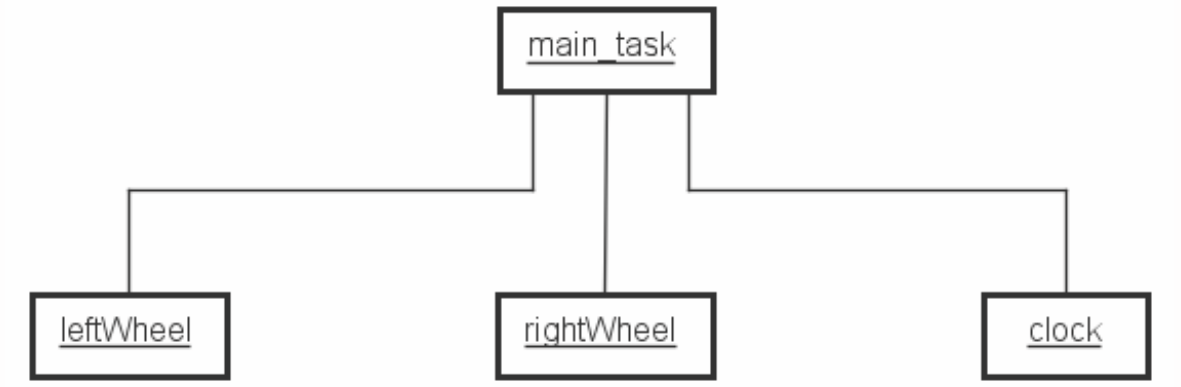

図 3. sample00のオブジェクト図

インスタンスをクラスと関係づける

それぞれのインスタンスと関連するクラスがわかるようクラスを指定してみましょう。

1. 「leftWheel」を選択します（4隅にハンドルが現れます）
2. 画面左下のプロパティエディタから「ベース」を選択し、「名前」が先ほど選択した「leftWheel」なのを確認します
3. ベースクラスのリストから「Motor - ev3api」を選びます
4. 「leftWheel」の図がMotorクラスのインスタンスとして表示されます
5. 同様にして「rightWheel」「clock」もクラスを関連づけます

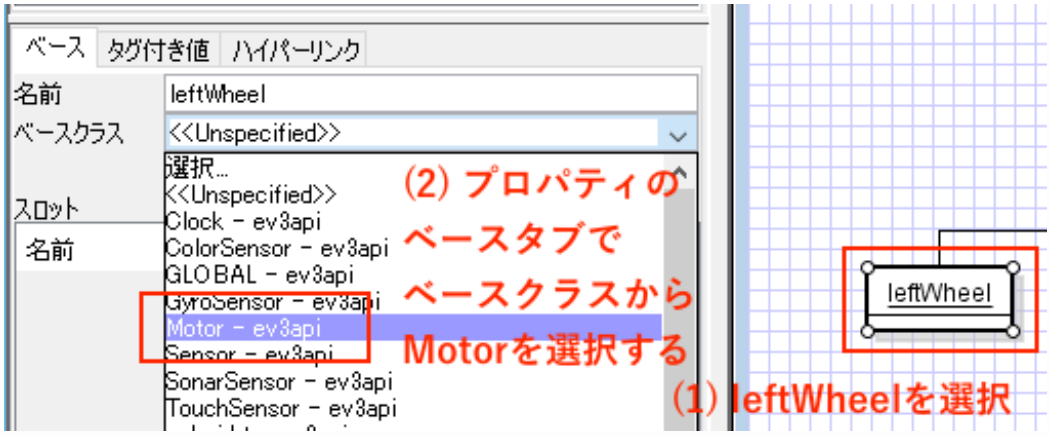

図 4. leftWheelにMotorクラスを割り当てる

作成したsample00のオブジェクト図２

クラスと関連づけたので、「オブジェクト名：クラス名」という表示に変わっています。

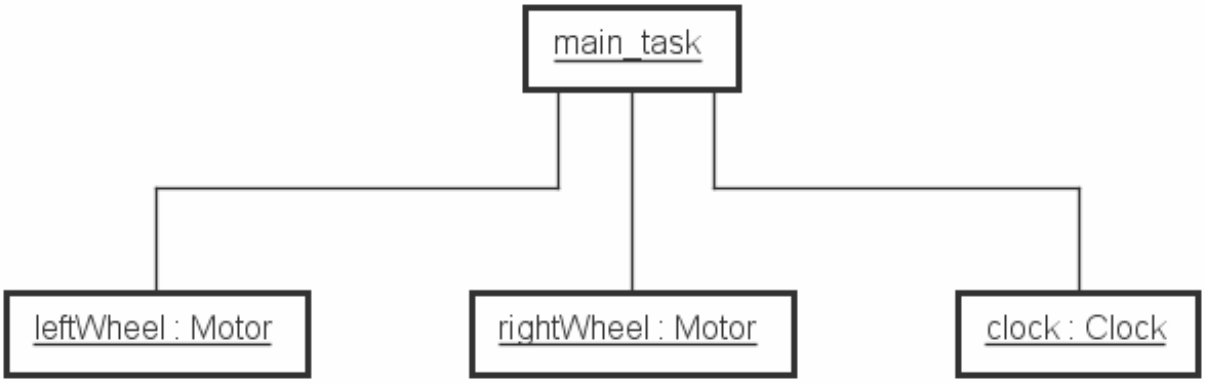

図 5. sample00のオブジェクト図（クラスと関連づけ後）

クラス図に表してみる

クラス図を作成して、インスタンス同士の関係をクラス同士の関係で表してみましょう。

1. 「図」→「クラス図」でクラス図を追加し、図の名前を「sample00のクラス図」にします
2. クラス図のパレットから「クラス」を選択し、アイコンが反転したら、マウスをダイアグラムエディタ上でクリックします
3. 「クラス0」を、sample00.cppで便宜上クラスとして扱うことにした「main_task」に変更します
4. 「構造ツリー」の「ev3api」パッケージを展開して「Motor」クラスをダイアグラムエディタにドラッグ＆ドロップします
5. 同様にして「Clock」クラスを配置します
6. 使っている側のクラスから、使われている側のクラスへ、参照に使っているインスタンス

ごとに単方向関連を引きます

7. 関連の端にマウスを移動すると「→」がポップアップするので、「leftWheel」などの関連端名をつけます
8. 「その他の表示/非表示」→「名前空間の表示」→「親の表示」を設定するとクラス名の前にパッケージの名前が表示されます

作成したsample00のクラス図

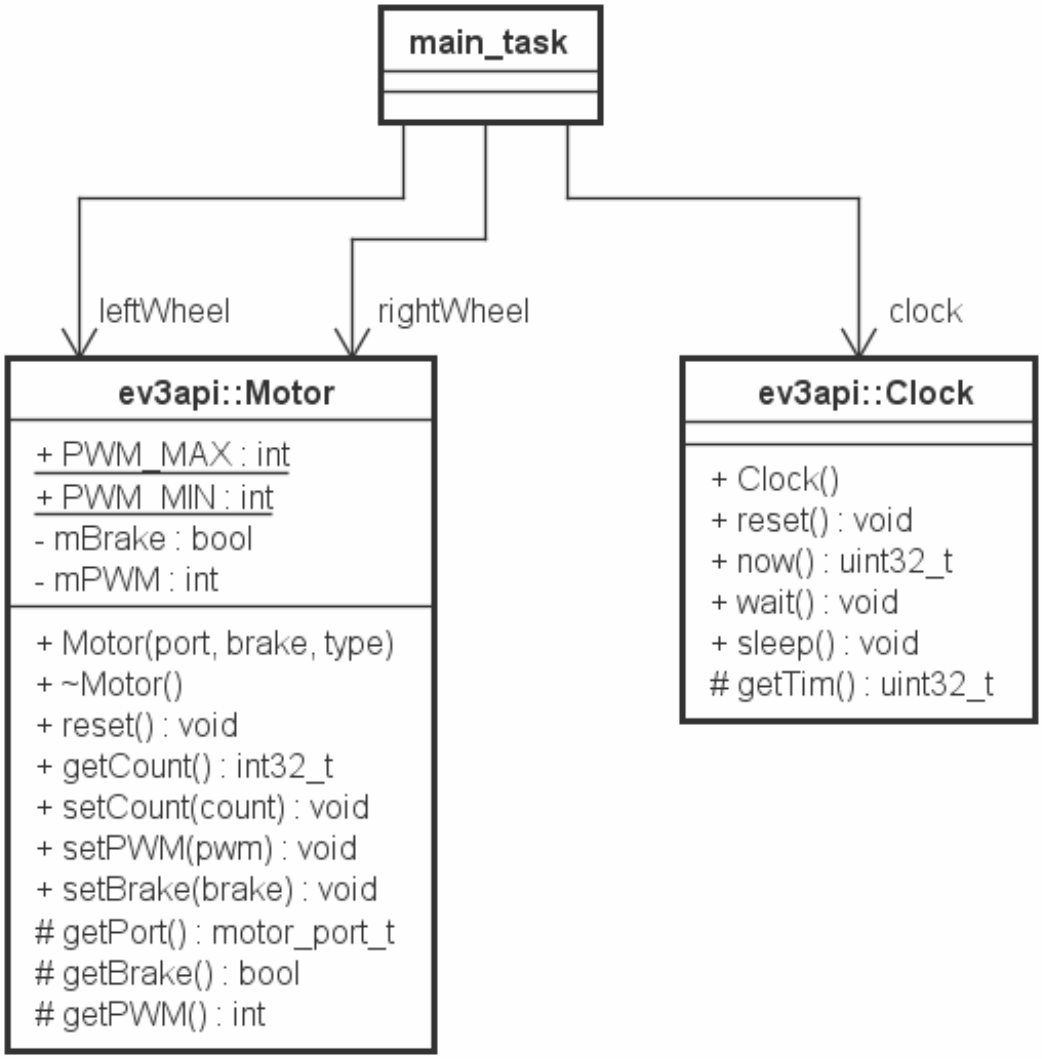

図 6. sample00のクラス図

作成した図から何が言えるでしょうか

- どんなクラスを使っているかわかりますか
- どんなクラスのインスタンス（オブジェクト）を使っているかわかりますか
- どんなことがやりたいシステムかわかりますか
- どんな処理をするシステムかわかりますか

sample00は…

MotorクラスとClockクラスを使っているアプリケーションであるというほかは、
どのようなことをやりたいのか書いていないということがわかりました。

システムの処理を担当するクラスを追加する

決まった走行（前進・後退を繰り返す）をロボットにさせたいのに、sample00にはそのことがわかるクラスがありません。

> このロボットがやる仕事を担当するクラスを作って、仕事の担当者としての名前をつけてみましょう。

クラス図にWalkerクラスを追加する

sample00のクラス図元に、Walkerクラスを追加し、runメソッドを追加したsample01のクラス図を作成しましょう。

1. 「ファイル」→「プロジェクトの別名保存」で「sample01」として保存します
2. クラス図の名前を「sample01のクラス図」に変更します
3. クラス図のパレットからクラスをドラッグ＆ドロップして、新しいクラス「Walker」を追加します
4. 「Walker」クラスにコンストラクタと「run」メソッドを追加します
    - デストラクタは作成せず、コンパイラに任せることにします
5. pwm、durationは「Walker」クラスの属性にします
6. 「main_task」は仕事を「Walker」に移譲するので、関連を引き直します
    - 「Motor」クラス、「Clock」クラスは「Walker」クラスと一緒に使うので、コンポジションにしておきます

作成したsample01のクラス図

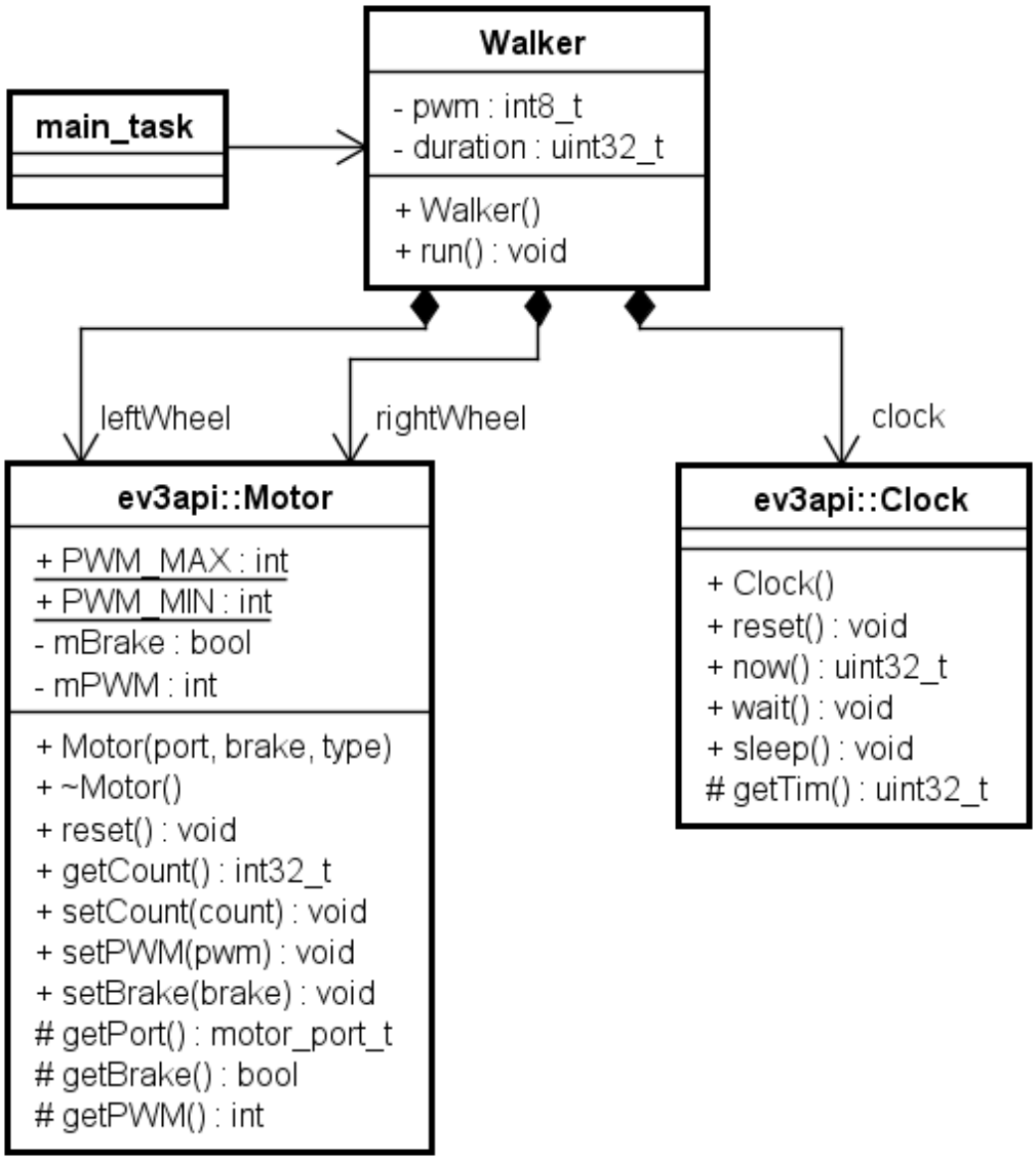

図 7. sample01のクラス図

sample01のコードを作成する

モデル図に合わせて、コードを変更しましょう。

1. サンプルコードの「sample00」ディレクトリをそっくりコピーして「sample01」ディレクトリを作りましょう
2. ファイルは分割しないで、「app.cpp」の中に「Walker」を作成しましょう
3. クラス図に従って「Walker」クラスを作成します
4. 「Walker」クラスの「run」メソッドに「main_task」の処理を移動します
5. 「main_task」は「Walker」クラスのインスタンスの作成と「run」メソッドの呼び出しを担当します

作成したコードは次ページ以降に掲載してあります。

コードが作成できたら、ビルドして、動作を確認しましょう。

sample01/app.cpp（その１）

```
#include "app.h"
#include "util.h"

#include "Motor.h"
#include "Clock.h"

using namespace ev3api;

class Walker {
public:
  Walker();
  void run();

private:
  Motor leftWheel;              (1)
  Motor rightWheel;             (1)
  Clock clock;                  (1)

  const int8_t pwm = (Motor::PWM_MAX) /
6;
  const uint32_t duration = 2000;
};
```

(1) MotorクラスとClockクラスのインスタンスは、Walkerのインスタンスと共に作成・破棄する

sample01/app.cpp（その２）

```
Walker::Walker():
  leftWheel(PORT_C), rightWheel(PORT_B) {
}

void Walker::run() {
  init_f(__FILE__);
  while(1) {
    msg_f("Forwarding...", 1);
    leftWheel.setPWM(pwm);
    rightWheel.setPWM(pwm);
    clock.sleep(duration);

    msg_f("Backwarding...", 1);
    leftWheel.setPWM(-pwm);
    rightWheel.setPWM(-pwm);
    clock.sleep(duration);

    // 左ボタンを長押し、それを捕捉する
    if (ev3_button_is_pressed(LEFT_BUTTON)) {
      break;
    }
  }
```

```
23
24    msg_f("Stopped.", 1);
25    leftWheel.stop();
26    rightWheel.stop();
27    while(ev3_button_is_pressed(LEFT_BUTTON)) {
28      ;
29    }
30  }
```

sample01/app.cpp（その３）

```
1  void main_task(intptr_t unused) {
2
3    Walker walker;                ❶
4
5    walker.run();                 ❷
6
7    ext_tsk();
8  }
```

❶ Waklerクラスのインスタンスを作成

❷ runメソッドを実行

ここまでのまとめ

sample01のコードとモデル図を作成しました。

この演習から何が言えるでしょうか。

- ロボットがやりたいことを担当するクラスを追加しました
  - 図やコードを読んだ時に、何がしたいのかわかるようになりました
  - やりたいことに「名前」がつきました
  - 「Walker」クラスの「run」メソッドに処理を走行の詳細を閉じ込めることができました
- 「main_task」は「Walker」クラスを使うだけになりました
- クラス図を変更し、それに合わせてコードを作成しました
  - クラス図とコードが対応していて、どちらで検討しても他方でも辻褄が合うようになりました

本資料について

資料名： コードとモデル図を対応づけてみよう （技術資料）

作成者： © 2016 by ETロボコン実行委員会
この文書は、技術教育「要素技術とモデルを開発に使おう」に使用するETロボコン公式トレーニングのスライドです。

2.0, 2016-05-21 12:56:41, 2016年用

2.0
Last updated 2016-05-21 04:05:05 JST