UECS-Pi SDK
デベロッパーガイド

バージョン 1.2

2016/05/11

【改定履歴】

版	改定内容	改定日
1.0	・初版作成	2014/11/03
1.1	・Raspberry Pi の I2C ポートの有効化方法を追記 ・その他誤記修正	2014/11/07
1.2	・ライブラリバージョンアップに伴う、サンプルコードの微修正 ・その他誤記修正	2016/05/11

目次

1. UECS-Pi SDK とは

1.1. 概要

UECS-Pi（ウエックスパイ）は、英国ラズベリーパイ財団によって開発された、ARM プロセッサを搭載したシングルボードコンピュータ「Raspberry Pi（ラズベリーパイ） Model B/B+/2」上で動作する、ユビキタス環境制御システム（UECS : Ubiquitous Environment Control System）実用通信規約 Ver1.00-E10（以下、E10 規格）準拠のソフトウェアです。

UECS-Pi SDK は、UECS-Pi のオープンソース版であり、E10 規格に準拠したソフトウェアを作成し、UECS 機器を自作するためのソフトウェア開発キットです。商用製品にも利用可能なオープンソースライセンスのライブラリを用いていますので、企業・個人に関わらず開発者は無償で利用できます。開発者は、UECS-Pi SDK で提供されるベースコードに、接続したいデバイスの制御コードと Web UI コードを追加で作成するだけで、短期間で E10 規格準拠の UECS 機器の作成が可能です。

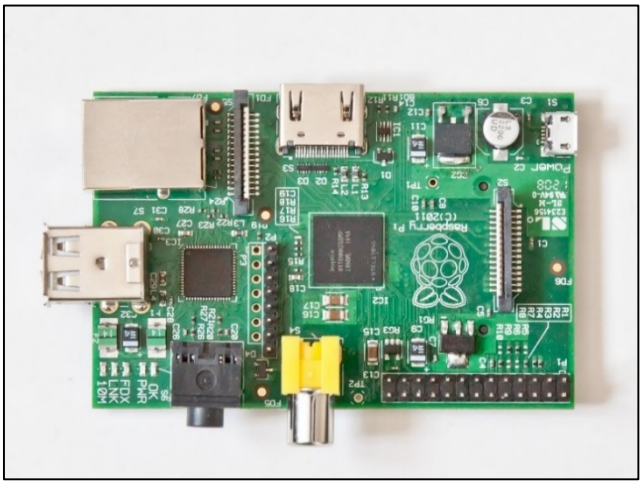


図 1：Raspberry Pi Model B

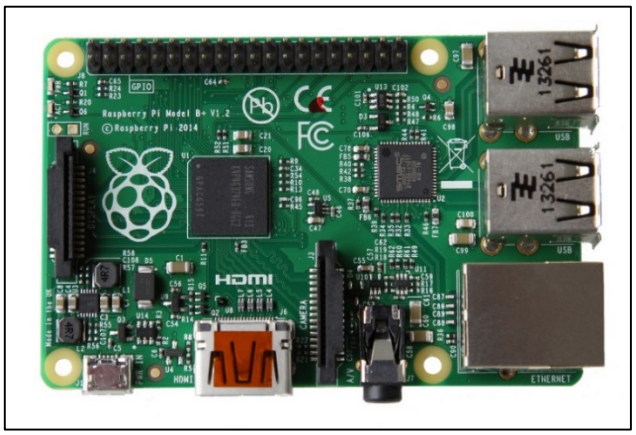


図 2：Raspberry Pi Model B+

1.2. UECS-Pi SDK の機能

UECS-Pi SDK は開発者に以下の機能を提供します。

No.	説明
1	E10 規格に則って動作するノードと CCM の基本ライブラリ
2	ノード設定、CCM 情報確認、ログ確認等が行える基本 Web UI 機能一式
3	Raspberry Pi に接続したデバイスを制御するための基本ライブラリ、フレームワーク
4	接続したデバイス用の Web UI を作成するための基本ライブラリ、フレームワーク

表 1：UECS-Pi SDK が提供する主機能

開発者はこれらの機能を用いて UECS 機器を開発します。例えば様々なセンサや、アクチュエータを接続して動作する UECS 機器を作成できます。UECS-Pi SDK では下図のように、Raspberry Pi 基板上の GPIO 端子や USB シリアル変換アダプタで接続されたセンサデバイスを UECS 仕様のセンサとして利用することが可能です。また GPIO 端子からアクチュエータに接続して、各種制御機器を動作させる事も可能です。

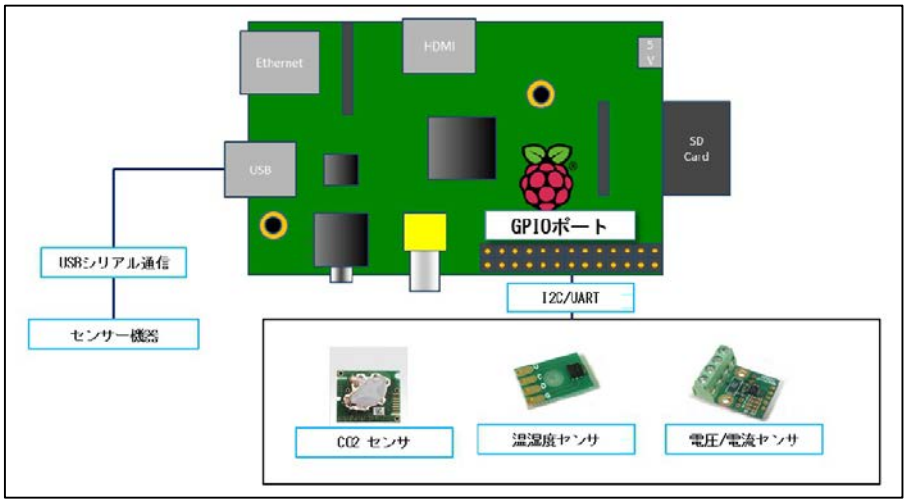


図 3：センサ接続イメージ

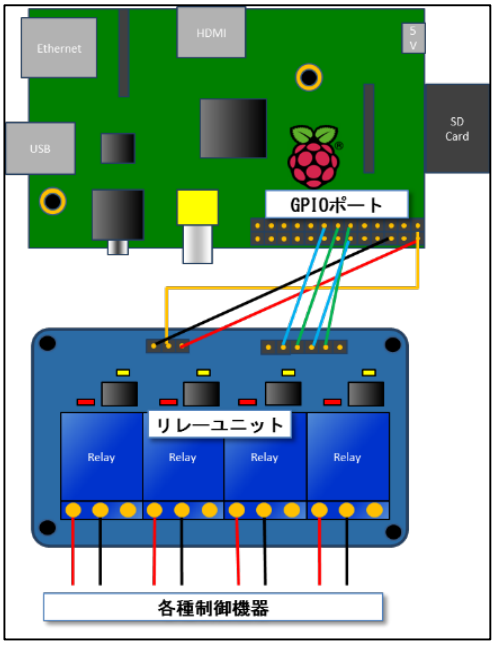


図 4：アクチュエータ接続イメージ

UECS-Pi SDK には、上図にあるようなデバイスを制御するための基本ライブラリ、フレームワークが用意されています。

Raspberry Pi に接続したセンサやアクチュエータの設定は、WebUI から行う事が可能です。WebUI も基本ライブラリ、フレームワークが用意されています。

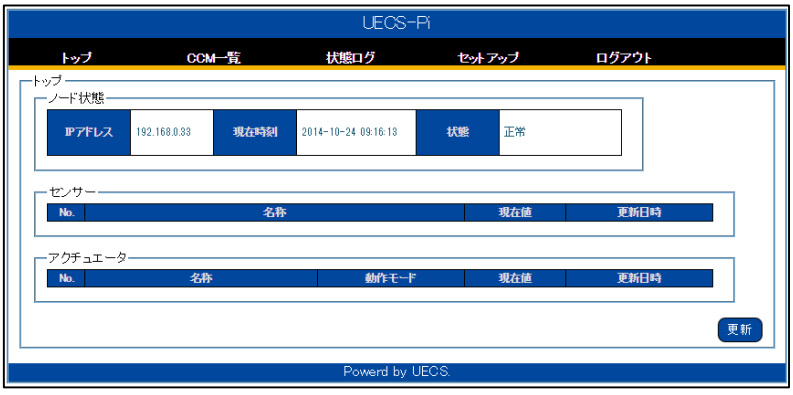


図 5：Web UI イメージ

UECS-Pi SDK を使って開発できる UECS 機器の全体動作イメージを示します。

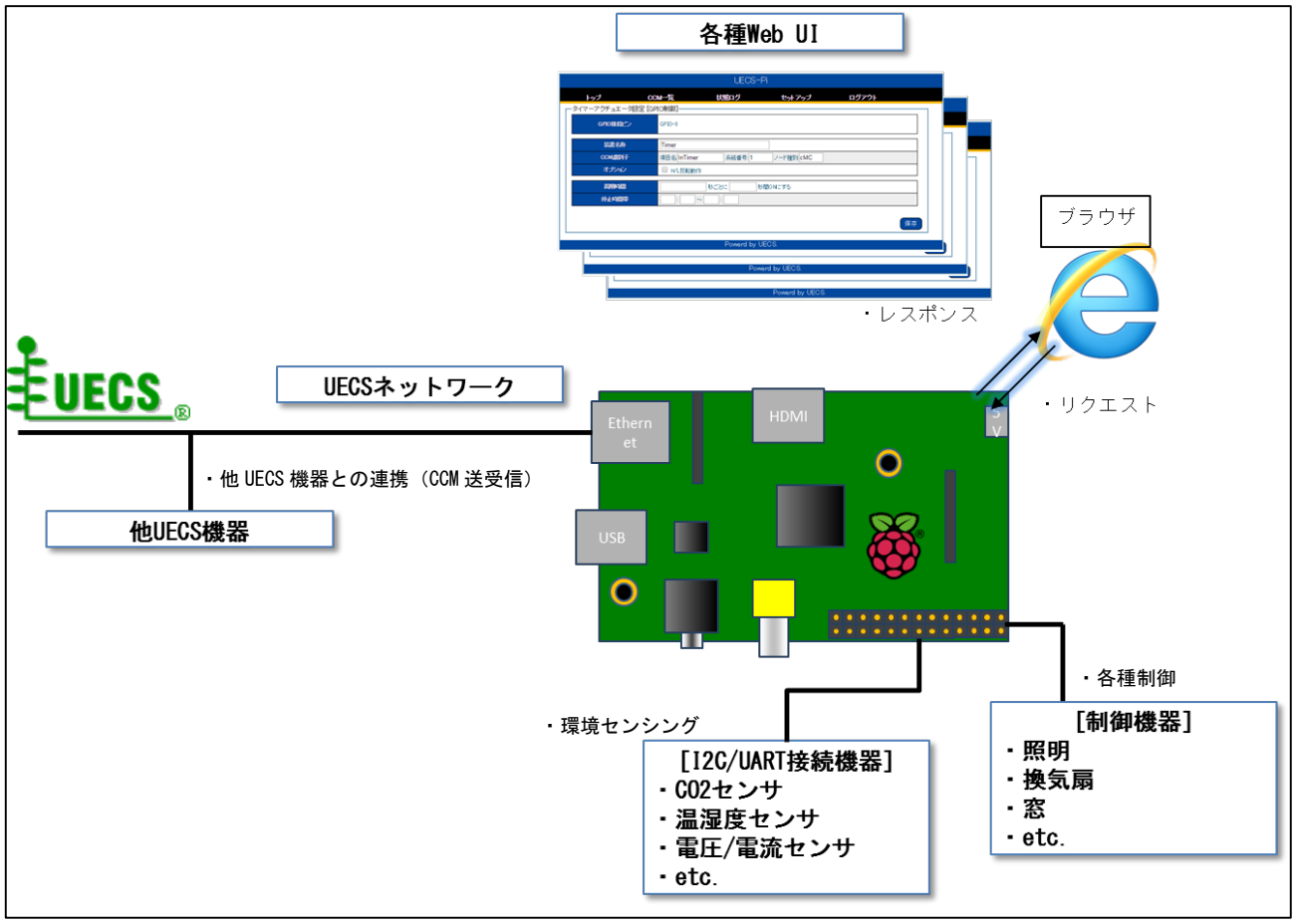


図 6：UECS-Pi SDK で開発する UECS 機器の動作イメージ

2. 開発環境セットアップ

2. 1. Eclipse セットアップ

本章では、Windows 環境での開発を例に説明します。Java が動作する Linux、Mac OS 等の環境でも開発可能です。

No.	名称	説明	ダウンロード URL
1	Pleiades	オープンソースの統合開発環境Eclipseに各種プラグインを統合した日本語版パッケージ	http://mergedoc.sourceforge.jp/

表 2：Eclipse セットアップに必要なソフトウェア一覧

① Pleiades の配布ページに移動し、Pleiades ALL in One ダウンロードから Eclipse 4.3 Kepler をクリックします。

図 7：Pleiades 配布ページ

② Java の Full Edition をダウンロードします。32bit か 64bit かは開発用 PC に応じて選択して下さい。

図 8：Pleiades ダウンロードページ

③ exe ファイルがダウンロードされるので実行します。下のポップアップが出た場合は「実行」ボタンを押して下さい。

図 9：exe ファイル実行時ポップアップ

④ Pleiades 関連ファイルの解凍先を尋ねられるので「参照」ボタンから指定します。指定を終えた場合か、初期の「C:¥pleiades」で良い場合は「解凍」ボタンを押すと指定フォルダにファイルが展開されます。

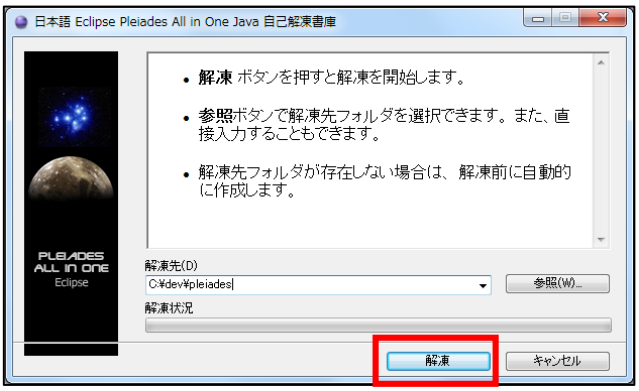


図 10：解凍画面

⑤ 解凍後 Pleiades の解凍先フォルダに移動し「eclipse」フォルダの「eclipse.exe」をダブルクリックします。

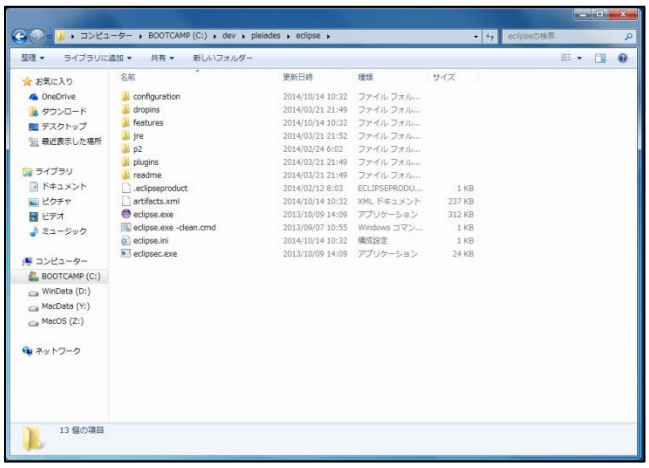


図 11：解凍先フォルダ画面

⑥　Eclipse 起動時に、ワークスペースを作成するフォルダを指定する画面が開きます。ワークスペースとは、Eclipse で作成したプロジェクトのリソースを格納するフォルダです。これは後で変更可能です。指定が終わったら「OK」ボタンを押すと Eclipse が起動します。

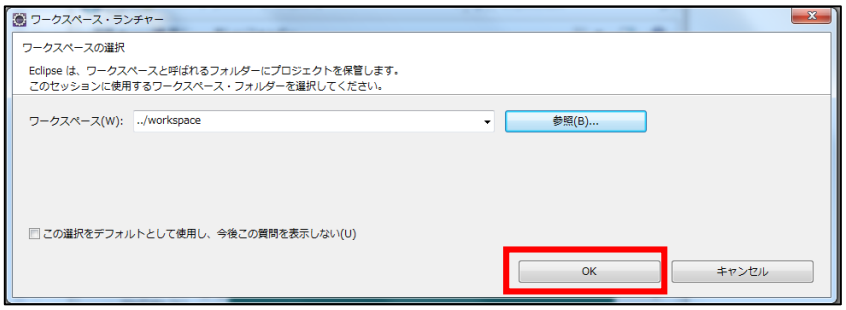


図 12：ワークスペースフォルダ指定画面

⑦　起動が完了すると以下の画面になります。これで Eclipse のセットアップは完了です。

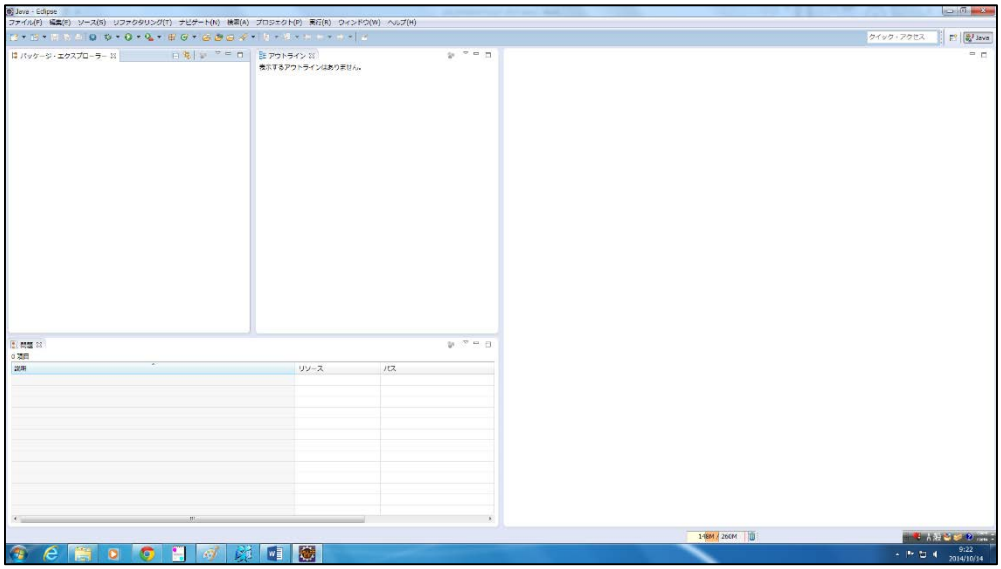


図 13：Eclipse 初期画面

2.2. Eclipse への UECS-Pi SDK セットアップ

No.	名称	説明	ダウンロード URL
1	UECS-Pi SDK	UECS機器開発のためのオープンソースSDK	http://www.wa-bit.com/business_info-2/smart-agri/uecs-pi

表 3：Eclipse への UECS-Pi SDK セットアップに必要なソフトウェア一覧

① UECS-Pi SDK の配布ページに移動し、zip ファイルをダウンロードして適当な場所に解凍します。

② Eclipse を開き、画面左上の「ファイル」メニューの中の「インポート」をクリックします。

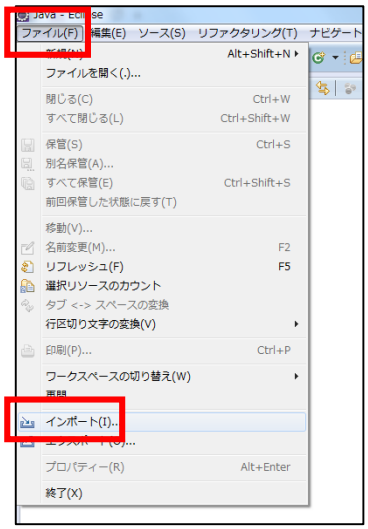


図 14：ファイルメニュー画面

③ 「一般」の中の「既存プロジェクトをワークスペースへ」をクリックし、「次へ」ボタンをクリックします。

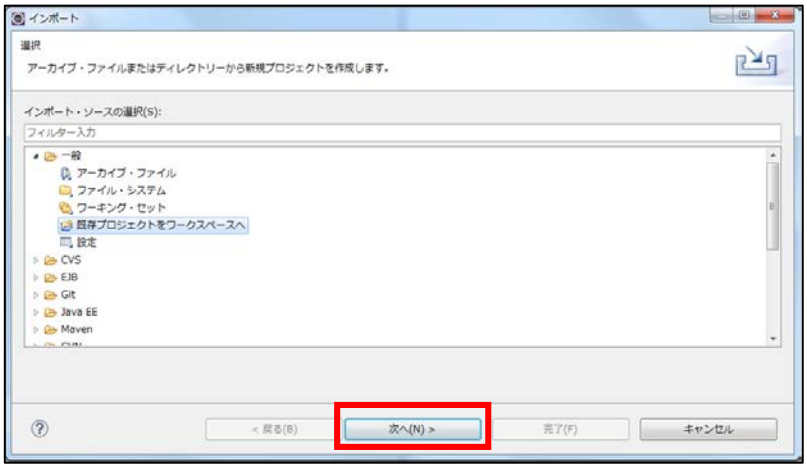


図 15：インポート画面 1

④ ルート・ディレクトリの選択で先ほど zip ファイルを解凍したフォルダ（UecsPi_SDK フォルダ）を選択し「完了」ボタンを押します。

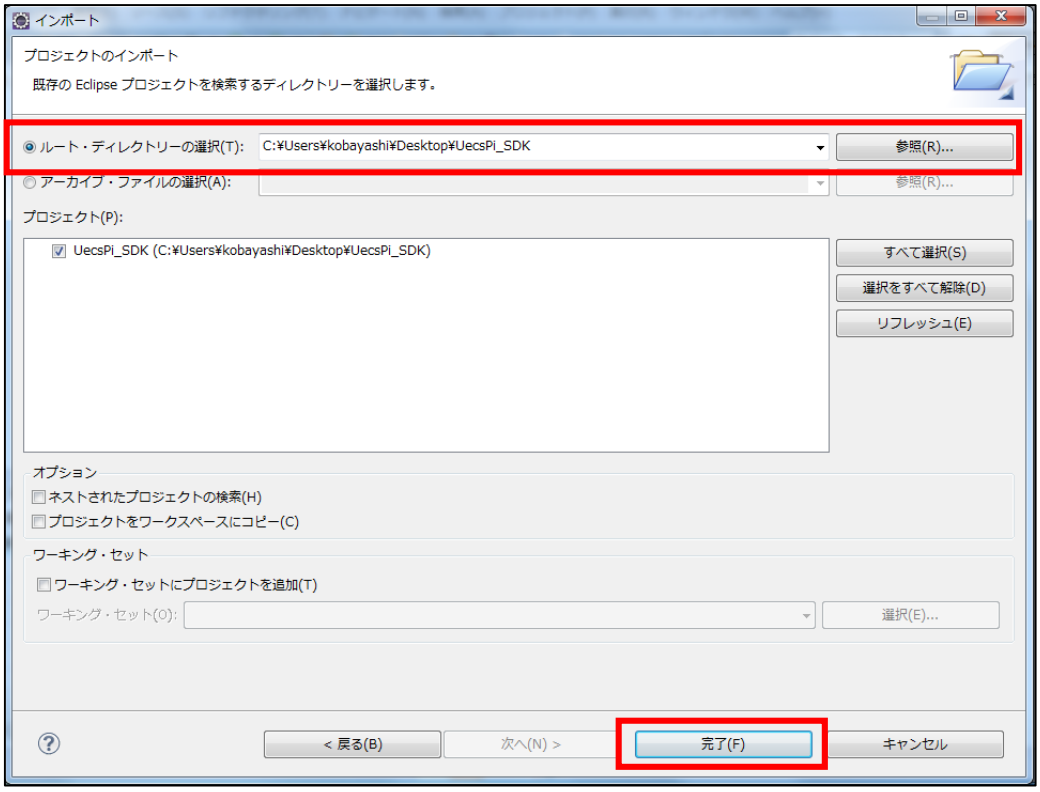


図 16：インポート画面 2

⑤ パッケージ・エクスプローラーの中に UECS-Pi SDK プロジェクトが追加されて表示されます。これで Eclipse への UECS-Pi SDK セットアップは完了です。この時点では Tomcat の定義がされていないのでエラーが出ています。

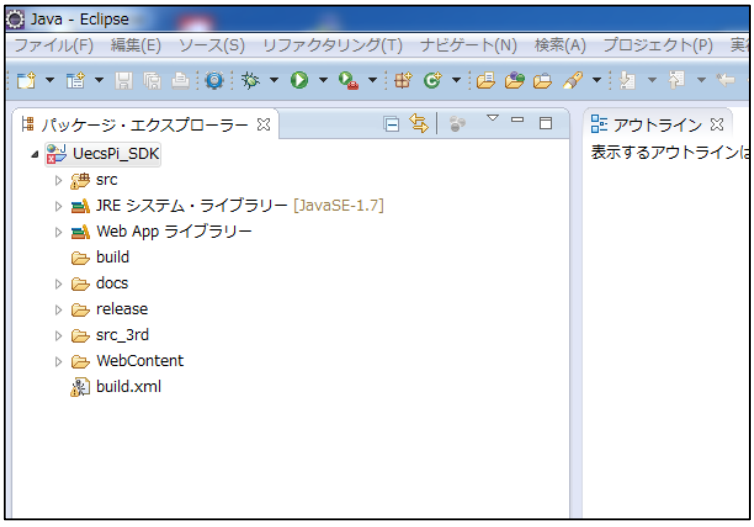


図 17：UECS-Pi SDK プロジェクト追加完了画面

2.3. Eclipse 上での Tomcat セットアップ

① Eclipse を開き「ウィンドウ」メニューの「設定」をクリックします。

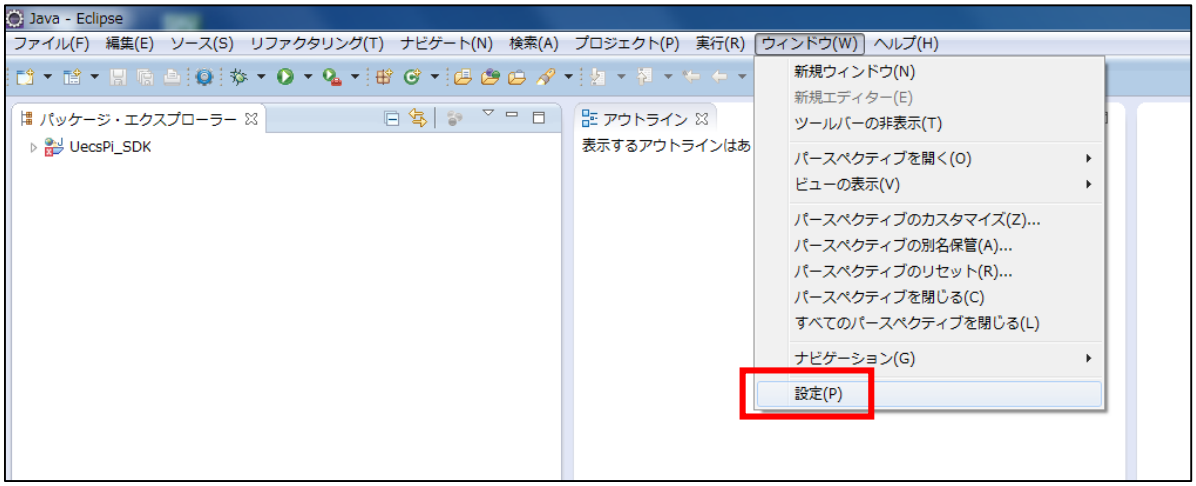


図 18：ウィンドウメニュー画面

② 設定ウィンドウが表示されるので「サーバー」の「ランタイム環境」を選択し「追加」ボタンをクリックします。

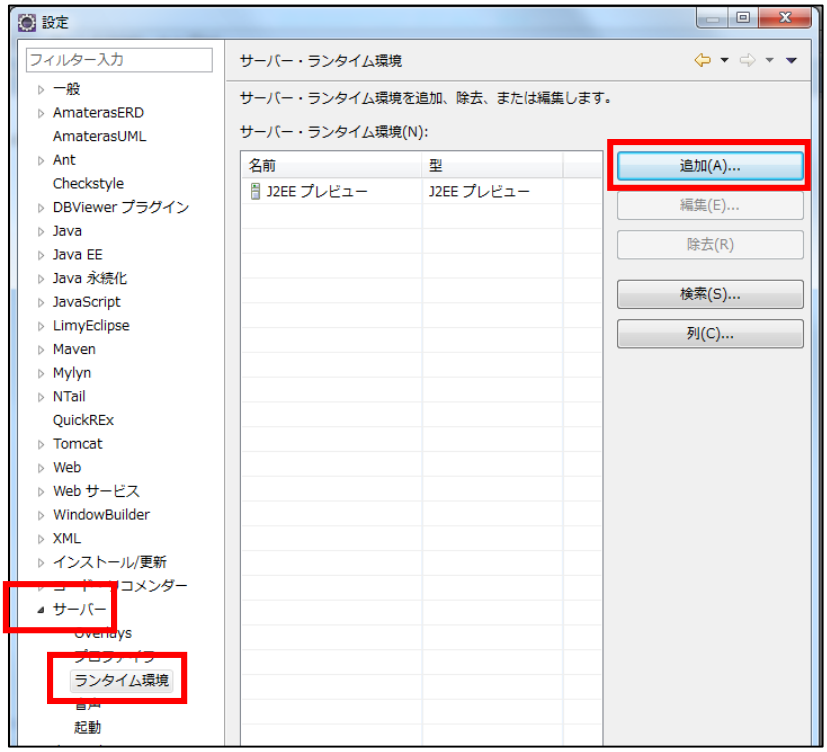


図 19：ランタイム環境設定画面 1

③ 「Apache」から「Apache Tomcat v7.0」を選択して、「新規ローカル・サーバーの作成」にチェックを入れたら「次へ」ボタンをクリックします。

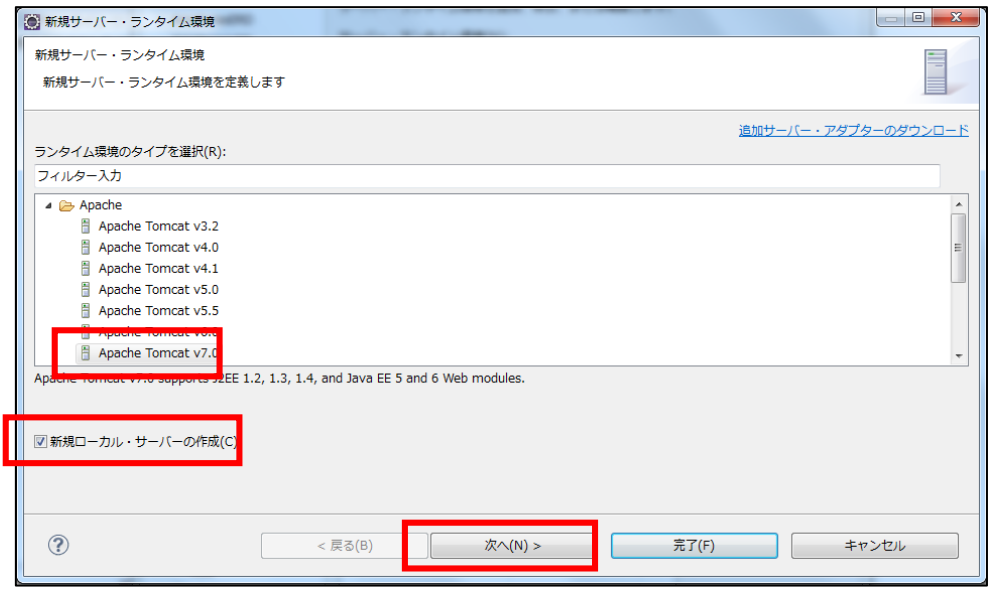


図 20：ランタイム環境設定画面 2

④ Tomcat インストールディレクトリを、Pleiades フォルダの「tomcat」フォルダ内にある「7」フォルダに設定し「完了」ボタンをクリックします。

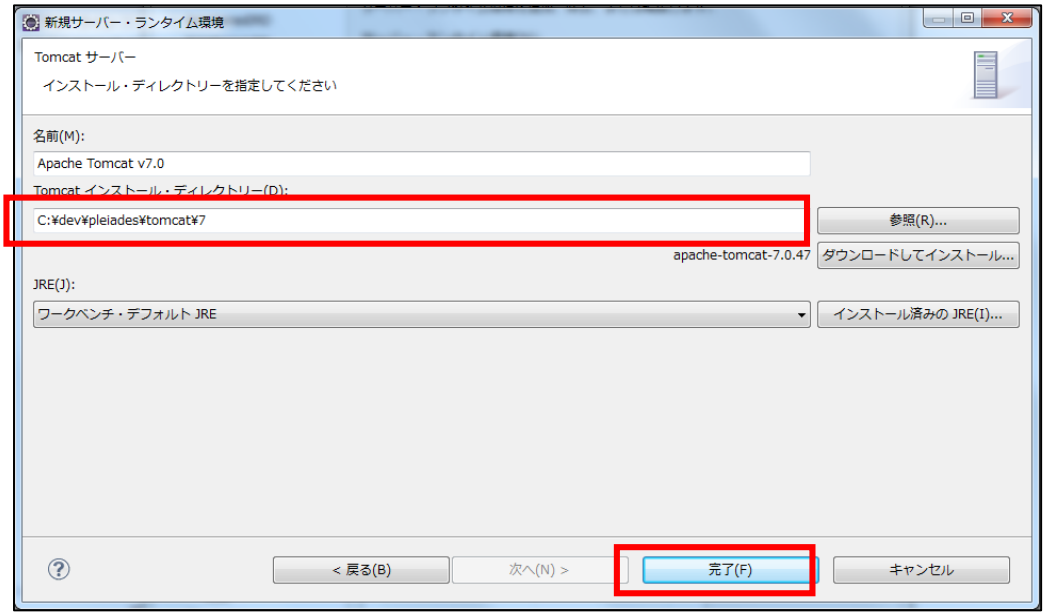


図 21：ランタイム環境設定画面 3

⑤ サーバが追加されるので選択して、「編集」ボタンをクリックします。

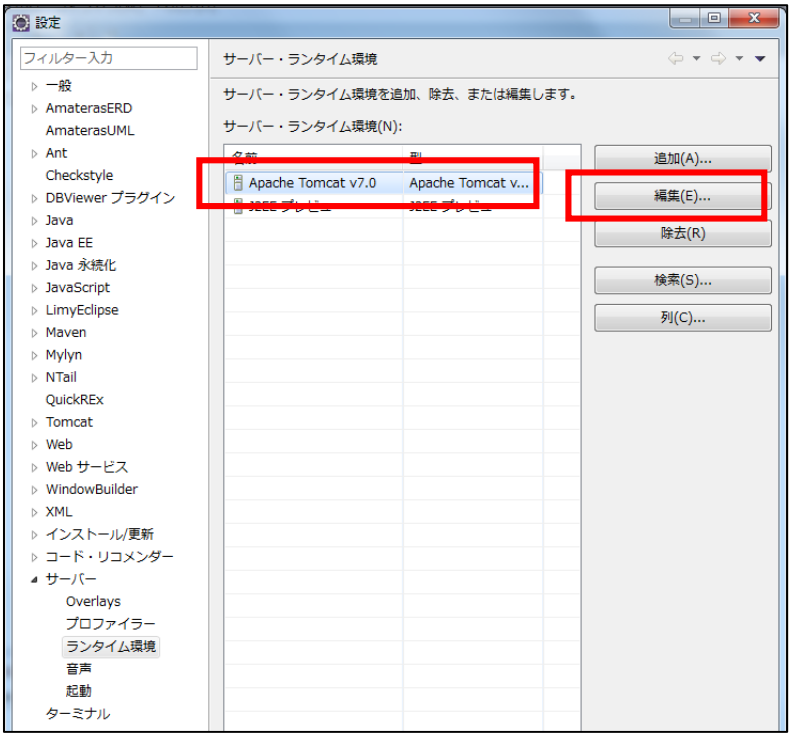


図 22：ランタイム環境設定画面 4

⑥　「JRE」を Java7 に変更し「完了」ボタンをクリックします。

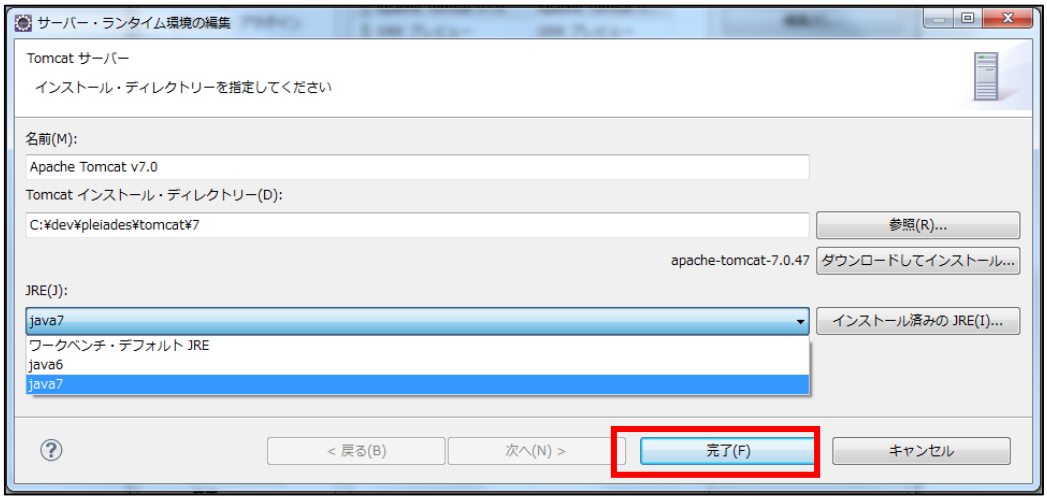


図 23：ランタイム環境設定画面 5

⑦　これでサーバの設定は完了です。「OK」ボタンをクリックします。

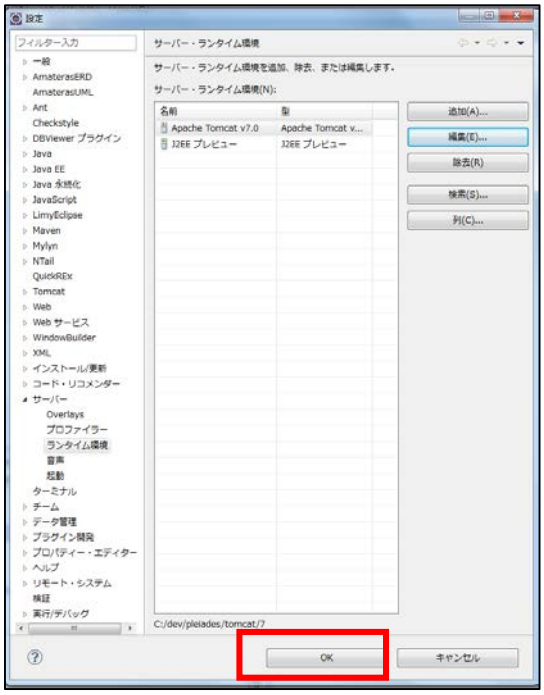


図 24：ランタイム環境設定画面 6

⑧　UecsPi_SDK プロジェクトを右クリックし「プロパティー」をクリックします。

図 25：プロジェクトメニュー画面

⑨　プロパティーから「Java コンパイラー」をクリックし、JDK 準拠を 1.7 に変更します。

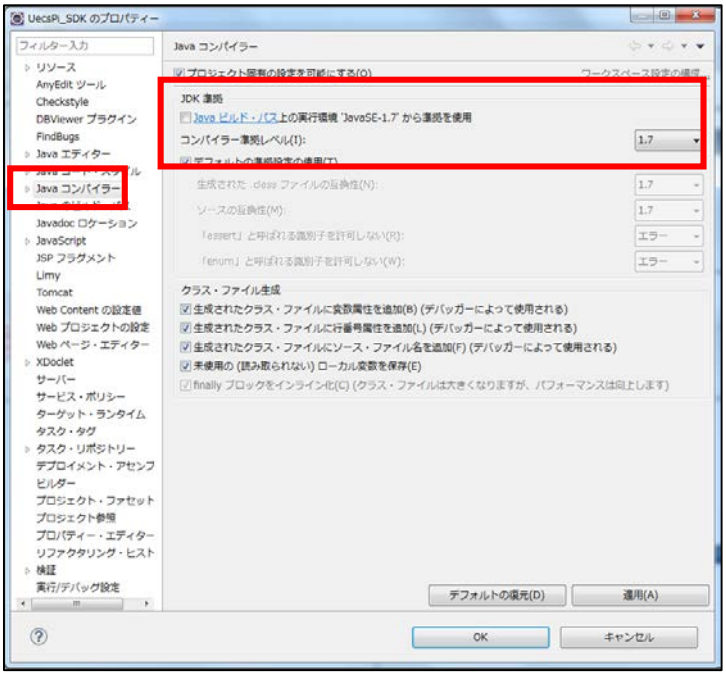


図 26：Java コンパイラー設定画面

⑩ 「サーバー」をクリックし、常に使用するサーバに作成したサーバを設定して「OK」ボタンをクリックします。これで Eclipse 上での Tomcat セットアップは完了です。

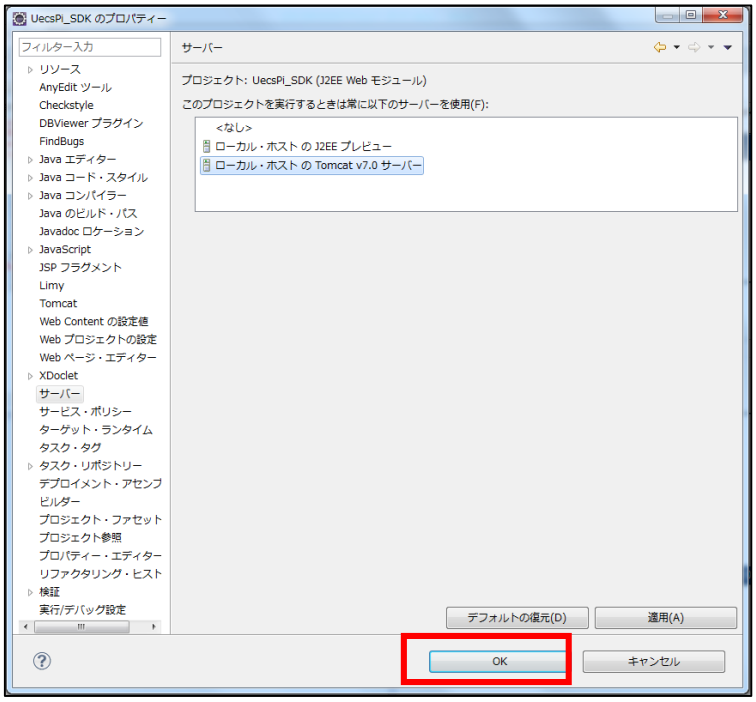


図 27：サーバー設定画面

2.4. Eclipse 上での UecsPi_SDK 実行

① UecsPi_SDK プロジェクトを Eclipse 上で実行したい場合は、プロジェクトを選択して「デバック」もしくは「実行」ボタンをクリックします。(初回起動時に実行方法選択ダイアログが表示されますので、「サーバーで実行」を選択してください)

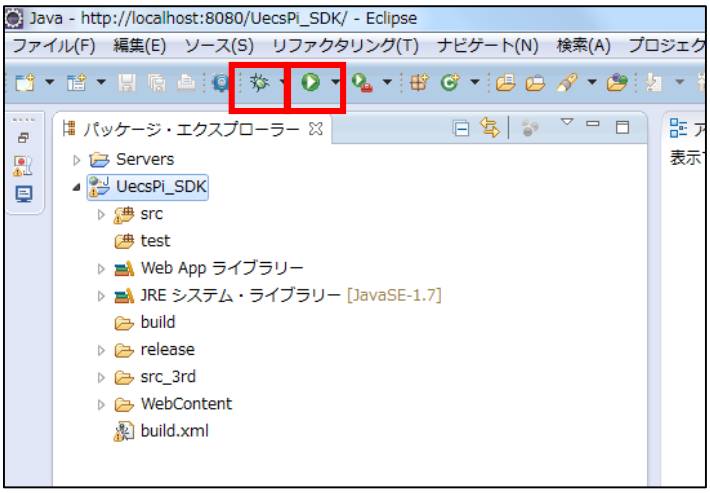


図 28：UecsPi SDK の実行

② 実行するとログイン画面が Eclipse 上に表示されます。「**admin**」と入力すればログイン出来ます。

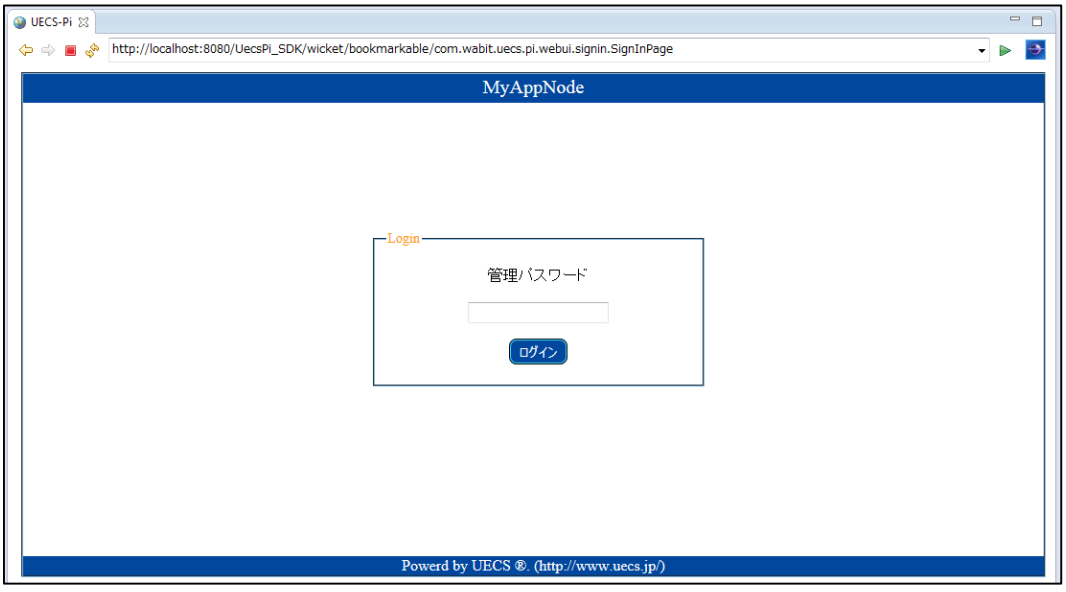


図 29：Eclipse 上でのログイン画面

Eclipse 上での UECS-Pi SDK 実行は、作成した Web UI コードなどの動作テストに使えます。開発 PC に接続できない機器との通信テストは Eclipse 上では行えませんので、Raspberry Pi 実機にインストールしてテストする必要があります。実機を使った実行方法は、「5. UECS-Pi アプリケーションインストール」以降で説明します。

3. 基本ライブラリ、フレームワーク概説

3. 1. 全体ライブラリ構成

UECS-Pi SDK の全体は、E10 規格をベースにした UECS 基本ライブラリをコアとし、そこに Raspberry Pi 上で UECS-Pi SDK が基本的な動作を行うためのライブラリ（以下、UECS-Pi 基本ライブラリ）をラップした構成になっています。UECS-Pi SDK を使用する開発者に提供されているのが、この 2 層構成ライブラリから成っているフレームワークです。開発者はそのフレームワークをベースにコードを作成する事となります。

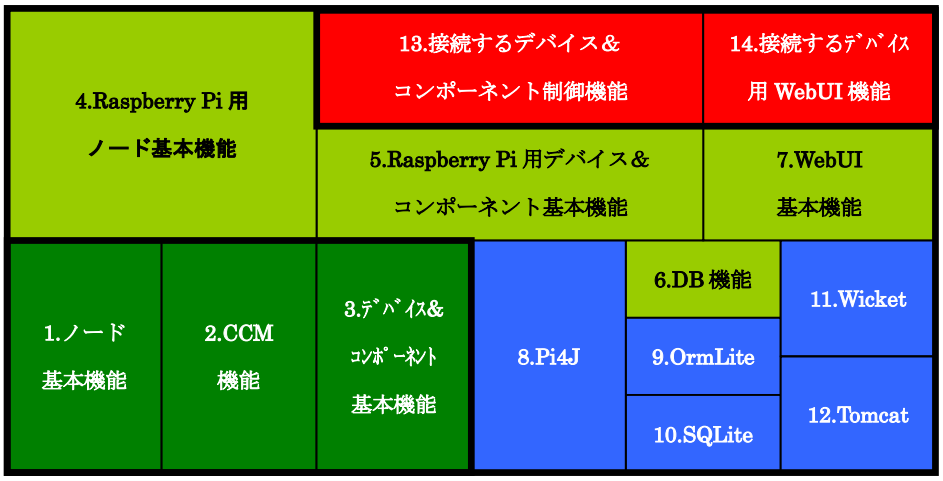


図 30：UECS-Pi SDK の全体ライブラリ構成

上図の 1～3 が UECS 基本ライブラリ、4～12 が UECS-Pi 基本ライブラリ、13～14 が開発者追加コードです。UECS-Pi 基本ライブラリのノード、デバイス、コンポーネント関連の機能は UECS 基本ライブラリを拡張して作成されています。また開発者が追加コードを作成する時は、基本的には UECS-Pi 基本ライブラリのクラスを拡張してコードを作成します。

開発者追加コードは大きく分けると「接続するデバイスを制御するコード」と「接続デバイス用の WebUI コード」の 2 種類あります。そのため UECS-Pi SDK では、デバイス接続用と WebUI 作成用の 2 つのライブラリを提供しています。

以下表に、UECS-Pi SDK を構成するライブラリの機能概要を示します。

No.	分類	名称	主な機能
1	UECS基本ライブラリ (uecs-core.jar)	ノード基本機能	ネットワーク通信、CCMサービス、デバイス管理機能
2		CCM機能	CCMサービス、各種CCMの定義
3		デバイス&コンポーネント基本機能	デバイス基本動作、コンポーネント基本動作

4	UECS-Pi基本ライブラリ (uecs-pi-core.jar)	Raspberry Pi用ノード基本機能	DB連携、DB登録済みデバイスの起動
5		Raspberry Pi用デバイス&コンポーネント基本機能	I2C、シリアル通信（UART、USB-シリアル）、GPIO接続を使った、デバイスとコンポーネントの基本動作
6		DB機能	ノード、デバイス、コンポーネント等のデータ処理
7		WebUI基本機能	UECS-Pi SDKの基本WebUI一式
8	UECS-Pi基本ライブラリ (サードパーティ製)	Pi4J	Raspberry Piに接続したデバイスをJavaから制御する 参照URL：http://pi4j.com/
9	UECS-Pi基本ライブラリ (サードパーティ製)	OrmLite	ORMマッパー 参照URL：http://ormlite.com/
10	UECS-Pi基本ライブラリ (サードパーティ製)	SQLite	軽量RDBMS 参照URL：https://bitbucket.org/xerial/sqlite-jdbc
11	UECS-Pi基本ライブラリ (サードパーティ製作)	Wicket	WebUI用のWebフレームワーク 参照URL：https://wicket.apache.org/
12	UECS-Pi基本ライブラリ (サードパーティ製)	Tomcat	WebUIやWicketが動作するHTTPアプリケーションサーバ 参照URL：http://tomcat.apache.org/
13	開発者作成コード	接続するデバイス&コンポーネント制御機能	開発者が接続するデバイス用の制御コード
14	開発者作成コード	接続するデバイス用WebUI機能	開発者が接続するデバイス用のWebUIコード

表 4：UECS-Pi SDK を構成するライブラリの機能概要

3.2. API ドキュメント(JavaDoc)、ソースコード

開発時に利用する主要なライブラリ群の API ドキュメントとソースコードは、UecsPi_SDK プロジェクト内の「src_3rd」フォルダ内に含まれています。必要に応じて、Eclipse の JavaDoc・ソースコードのバインド機能を用いて参照することが可能です。また、ライブラリ自体を自由にカスタマイズすることも可能です。

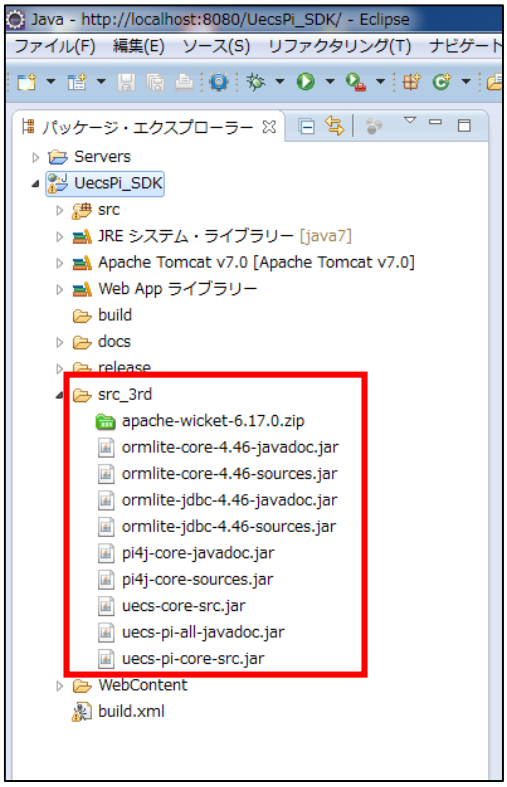


図 31：JavaDoc/ソースコード jar

3.3. ノード、CCM 送受信機能

3.3.1. ノード基本機能

UECS では、制御したい環境内にコンピュータとセンサやアクチュエータを設置し、イーサネットで接続します。この機器は「ノード」と呼ばれます。ノード同士は、E10 規格に則った XML データの相互通信を行い、計測や制御を自律的に行います。この XML データは「UECS 共用通信子（UECS-CCM）」と呼ばれます。以下に UECS の模式図を示します。

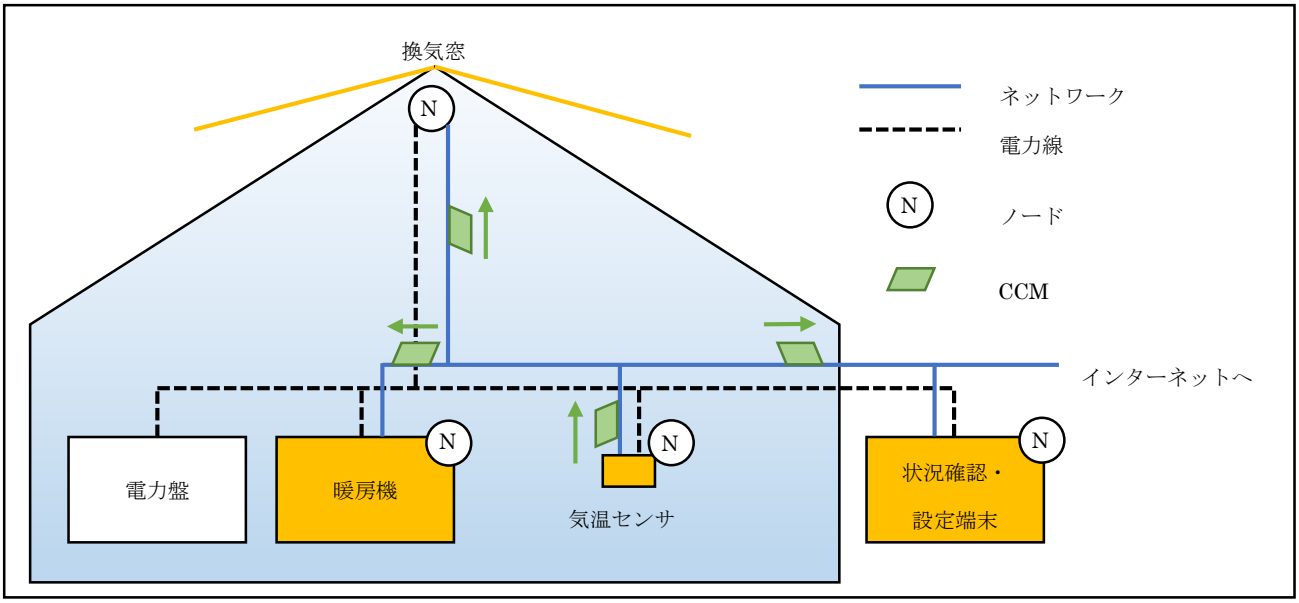


図 32 ：UECS ノードが配置されたガラスハウスの模式図

※UECS についての詳細はユビキタス環境制御システム研究会 HP（http://www.uecs.jp/）を参照して下さい。

※CCM についての詳細は UECS 通信実用規約 1.00-E10（http://www.uecs.jp/uecs/kiyaku/UECSStandard100_E10.pdf）を参照して下さい。

例えば上図では、気温センサから CCM が流れ、暖房機のノードがそれを受け取って点火したり、換気窓が閉じたり、状況確認・設定端末に気温が表示されるといった事が可能です。

UECS 基本ライブラリの中心になっているのが、この機能を実装した CCM 機能です。CCM 機能は、内部で CCM 受信と CCM 周期送信を行います。この機能により UECS-Pi SDK は他 UECS 機器と連携を行います。

また UECS-Pi SDK のノードは、ノード自身のセットアップ（ホストの IP 設定、ノード状態通知 CCM(cnd.kNN)登録、ブロードキャストアドレスの設定）や、DB に登録されているデバイス起動を行います。

3.3.2. CCM と CCM サービス

CCM 機能の周期送信処理では CCM が周期的に送信されます。複数の CCM を送信する事も可能です。ノードは「CCM サービス」を使い、CCM を送信します。CCM サービスとは、CCM の状態を管理するサービスクラスです。ノード、CCM サービス、CCM の関係は下図のようになっています。

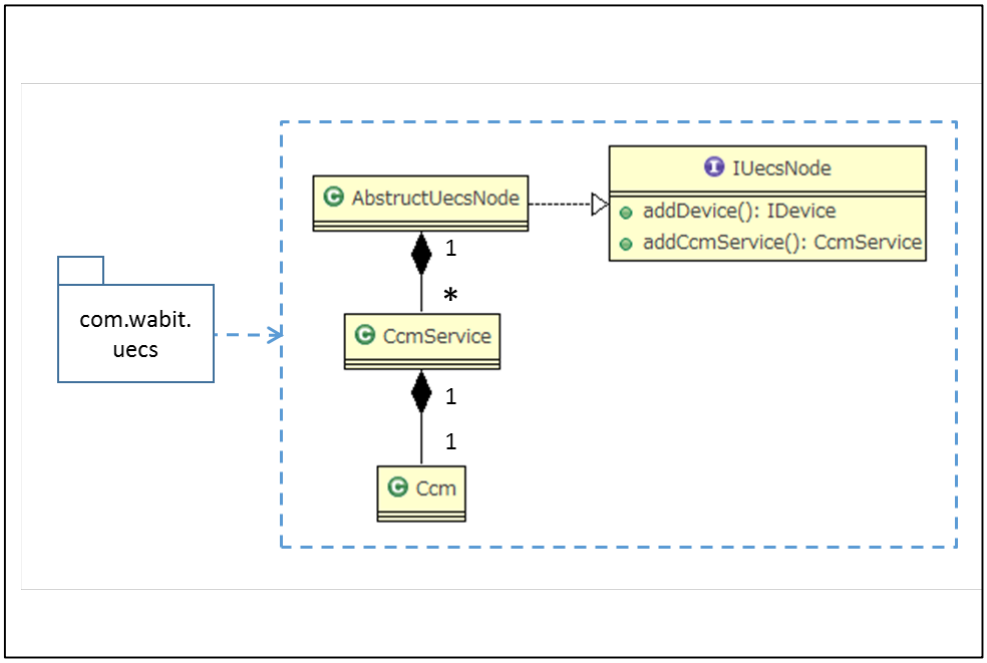


図 33：ノード、CCM サービス、CCM の関係

Ccm クラスの派生クラスが、com.wabit.uecs.ccm、com.wabit.uecs.protocol パッケージに含まれていますが、AbstractUecsNode 内部で自動的に利用されますので、開発者が通常のアプリケーション開発で直接これらの Ccm 派生クラスを意識する必要はありません。

3.4. デバイス、コンポーネント機能

3.4.1. ノード、デバイス、コンポーネントの概念

UECS-Pi SDK にデバイスを接続する際は、デバイスだけでなく、ノードとコンポーネントを考えます。ノード、デバイス、コンポーネントは概念です。1 つのノードには複数のデバイスを所属させられ、1 つのデバイスには複数のコンポーネントを所属させられます。

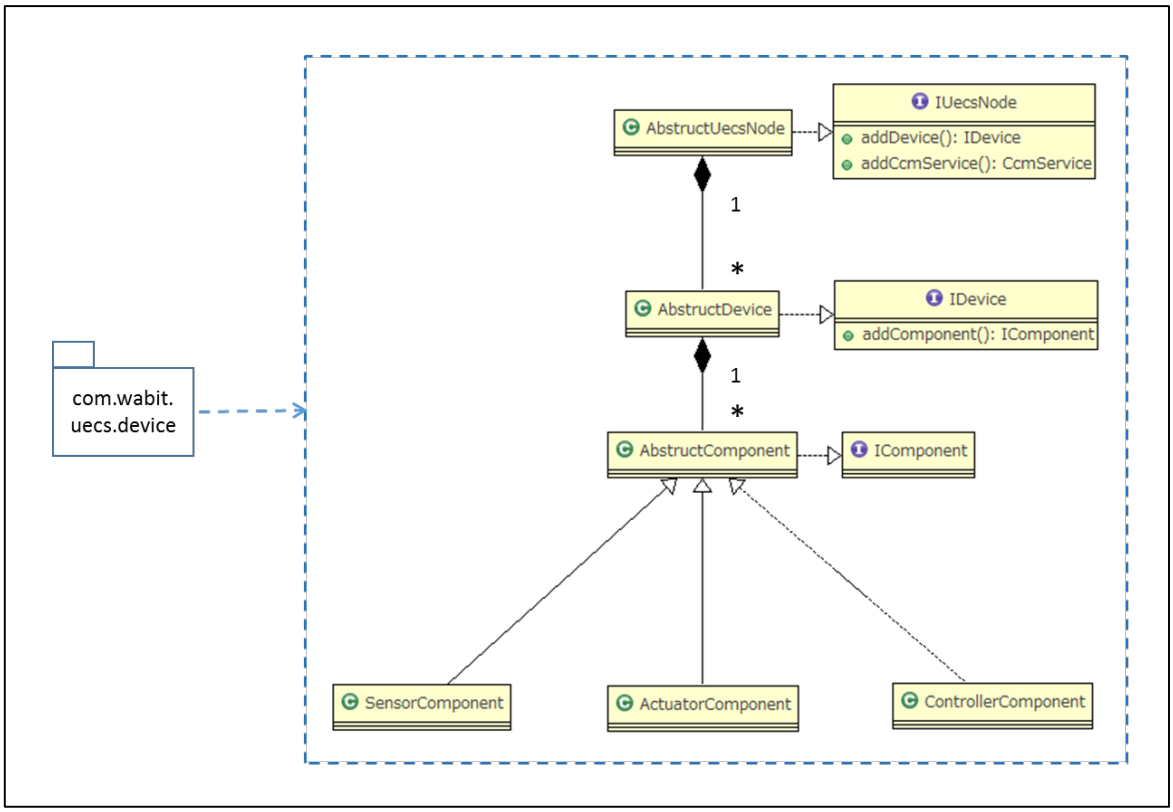


図 34：ノード、デバイス、コンポーネントの関係

例えば電流･電圧･電力計測が出来る INA226 モジュールを Raspberry Pi に接続して UECS-Pi SDK として動作させるとすると、見た目の上では Raspberry Pi がノード、INA226 がデバイス、INA226 に付いている電流・電圧・電力を測るそれぞれのセンサがコンポーネントとなります。

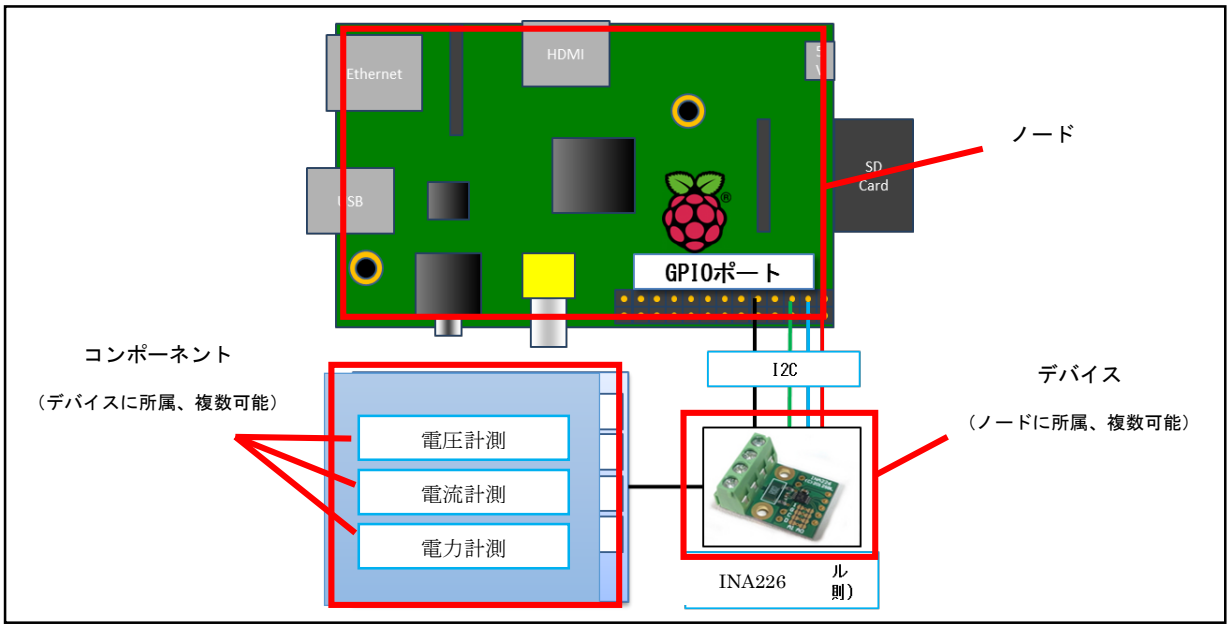


図 35：INA226 接続イメージ

配布直後の UECS-Pi SDK は何も接続されていないノードとなっています。開発者はそこに様々なデバイスを接続し、デバイスとデバイスに関連付くコンポーネントのプログラムを作成する事となります。

3.4.2. ノード、デバイス、コンポーネントの設定ファイル

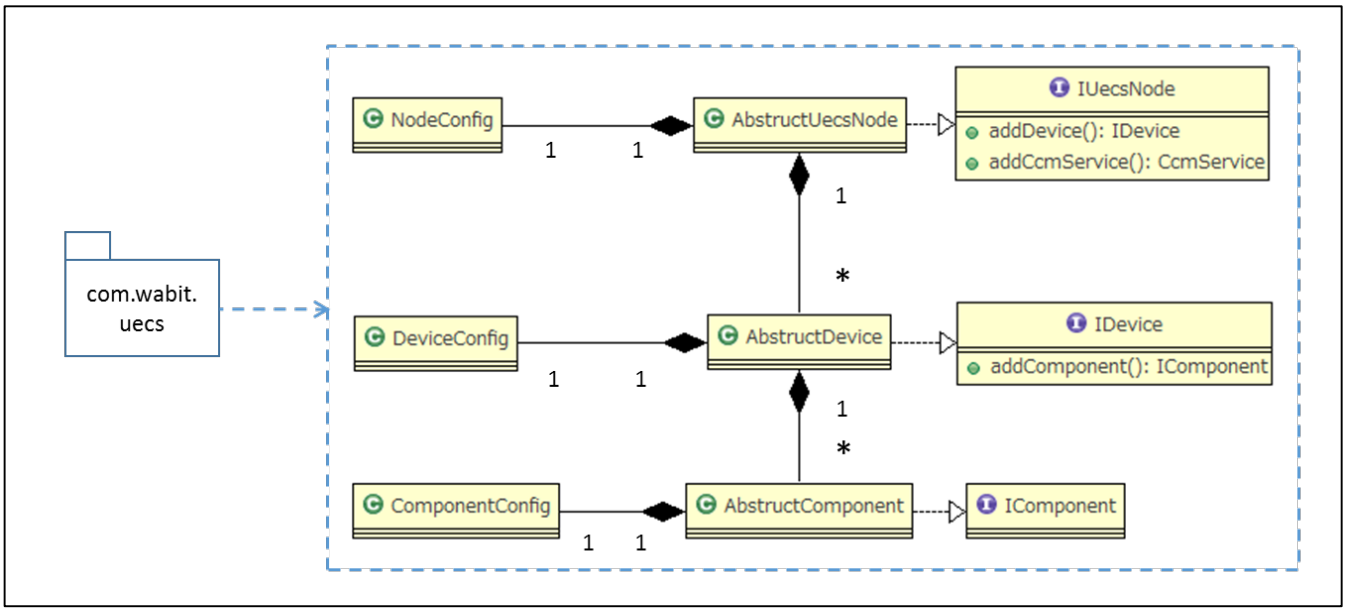


図 36：ノード、デバイス、コンポーネントと各設定ファイルの関係

ノード、デバイス、コンポーネントは、各々設定ファイルを持っています。設定ファイルでは定数の設定を行い、ノード、デバイス、コンポーネントの各ファイルでそれを利用してプログラムを記述しています。UECS-Pi SDK では設定を key / value の形で扱っています。プログラムを記述する側のファイルでは、設定ファイルに記述された key を呼び出すことで、その key に関連付く value を利用します。

3.4.3. デバイス通信・制御のライブラリ

UECS-Pi SDK のフレームワークに用意されているデバイス用の各種通信方式は I2C、シリアル通信（UART、USB-シリアル）です。また、GPIO 接続によるアクチュータ制御用のライブラリも用意されています。開発者は各方式の基本ライブラリクラスを拡張して、接続するデバイス制御用のオリジナルクラスを作成します。そこに接続するデバイスに関連付くコンポーネント用のクラスを作成する事で、UECS-Pi SDK に接続するデバイスとコンポーネントを動作させる準備が出来ます。これらのライブラリクラスは、UECS 基本ライブラリの派生クラスとして、com.wabit.uecs.pi パッケージ以下に含まれています。また、開発した UECS 機器の上でデバイスやコンポーネントの設定値をユーザーに変更させる必要がある場合は、WebUI の作成を行う事になります。（チュートリアルの章で詳細を説明します）

3.4.4. コンポーネントと CCM

コンポーネントをノード内で動作させると、必ず対応する CCM も動作します。コンポーネントが起動する時に、自らに関連付く CCM サービスをノードに登録するためです。1 つのコンポーネントに所属する CCM の数は、コンポーネントの種類によって変わります。

コンポーネントの種類は、センサ、アクチュエータ、コントローラの 3 種類です。以下にそれぞれの機能を示します。

クラス分類	機能	登録される CCM の種類
SensorComponent の派生クラス	センサー機器と直接通信してCCM送信、あるいは他のUECS センサーノードからのCCM受信でセンサー値を受信する。	DataCcm（受信/送信）
ActuatorComponent の派生クラス	リレー制御やモーター制御により、接続された制御機器を動作させる。ノード自身の自律動作以外に、コントローラからの遠隔制御CCMを受信して動作する。	OprCcm（送信側） RcACcm（受信側） RcMCcm（受信側）
ControllerComponent の派生クラス	アクチュエータとネットワーク経由でCCMのやり取りを行って遠隔コントロールする。 ※例えばツマミがひねられたらコントローラがCCMを送信するようにしておき、アクチュエータ側にそのCCMが来たら機器を作動させるプログラムを入れておけば、間接的な機器制御が行える。	OprCcm（受信側） RcACcm（送信側） RcMCcm（送信側）

表 5：各コンポーネントの機能

これらの CCM がコンポーネントによってノードに登録される事により、ノードの周期送信機能がそれらの CCM を送信する事や、WebUI の CCM 一覧確認画面からそれらの CCM を確認する事が可能になります。

3.4.5. ノード、デバイス、コンポーネント、CCM の動作

これまでに紹介したノード、デバイス、コンポーネント、CCM の関係を図示します。

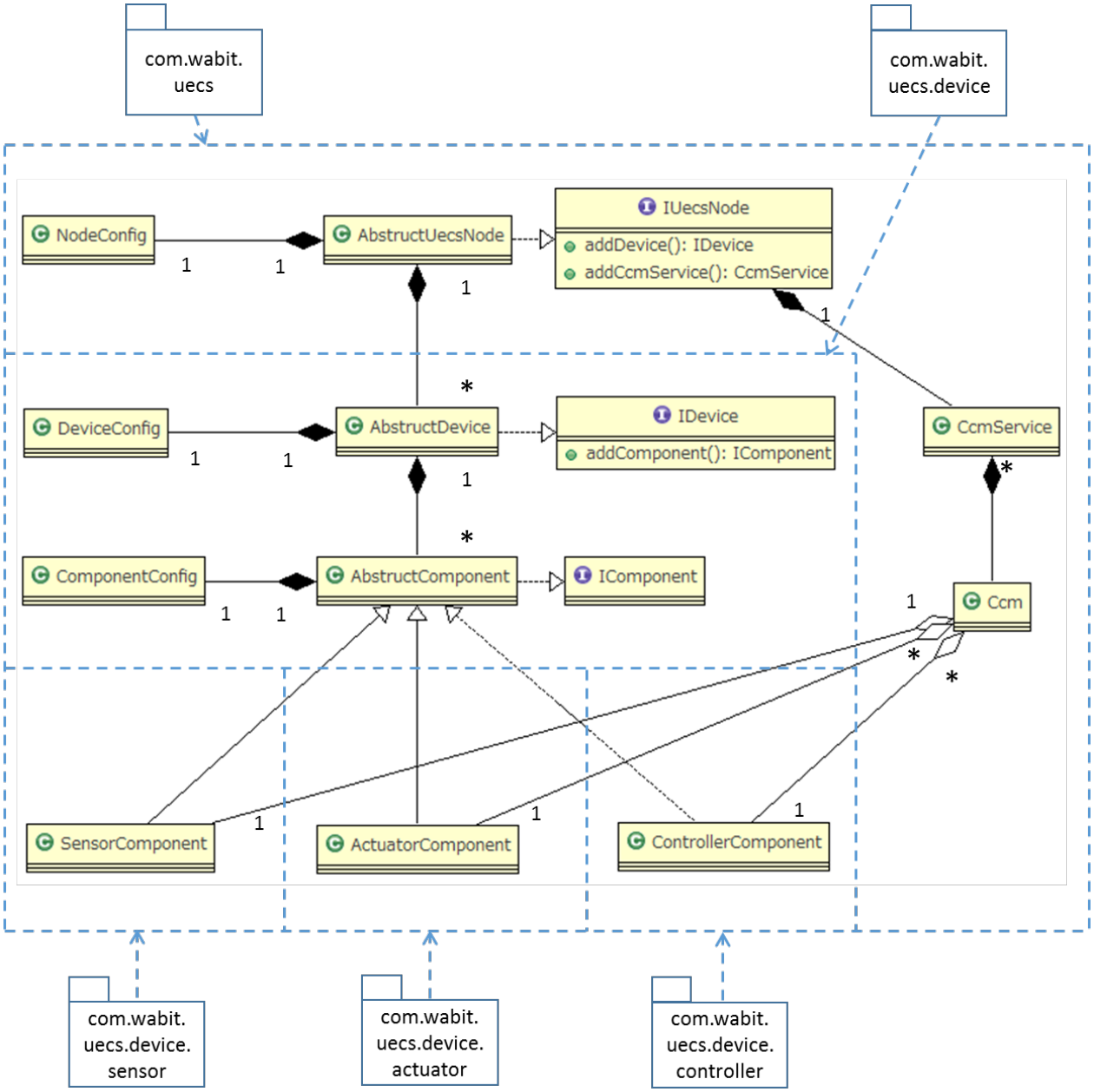


図 37：ノード、デバイス、コンポーネント、CCM の関係図

また、ノード、デバイス、コンポーネント、CCM の動作は一連の流れになっています。UECS-Pi SDK の起動プロセス～定常動作にそれが現れています。以下に図示します。

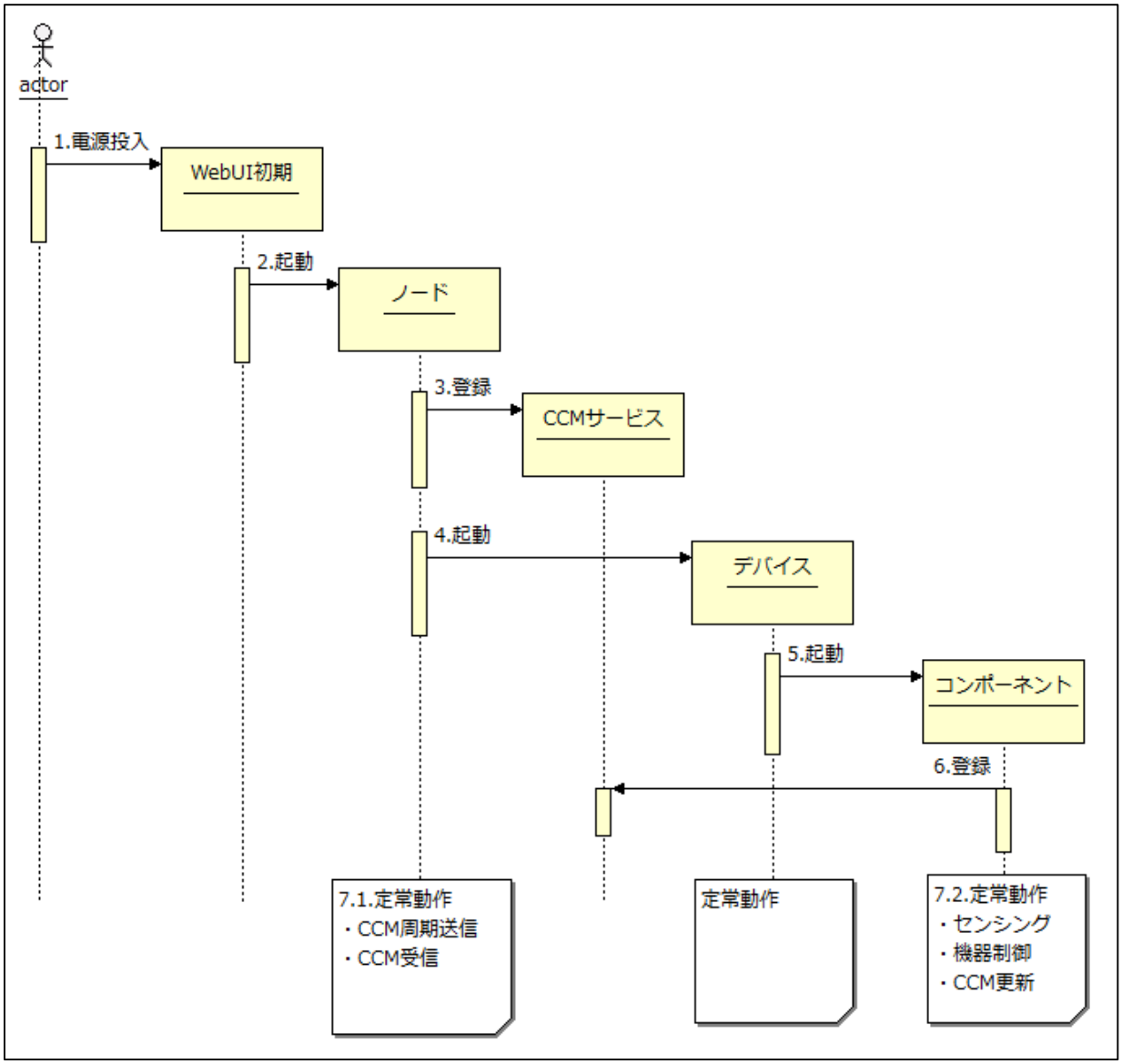


図 38：UECS-Pi SDK 起動プロセスシーケンス

順序	動作名称	動作内容
1	電源投入	電源投入され、OS→Tomcatが起動。Tomcatはweb.xmlで定義されている WebUI初期処理実行クラスを起動
2	WebUI初期処理実行クラス起動	WebUI初期処理実行クラスはノードとWebUIトップ画面を起動
3	ノード起動1	ホストIP設定、ノード用CCM登録、ブロードキャストアドレス設定を行う
4	ノード起動2	ノードが所属するデバイスを起動
5	デバイス起動	各デバイスが所属コンポーネントを起動
6	コンポーネント起動	各コンポーネントがCCMを作成し、CCMサービスの形でノードに登録する
7.1	ノード定常動作開始	ノードが定常動作開始。CCMサービスに登録済のCCMの周期送信と、CCM受

		信を行う
7.2	コンポーネント定常動作開始	コンポーネントが定常動作開始。何か変化があった時にデータ（センサの取得値や、アクチュエータの動作値）を更新し、同時に所属するCCMの値も更新する

表 6：UECS-Pi SDK 起動プロセス解説

またノード再起動の時は、それまで登録されていたデバイスや CCM サービスはリセットされ、改めて DB からデバイスやコンポーネントが呼び出されて動作を開始します。

3.4.6. Raspberry Pi へのデバイス接続方法

センサデバイスを Raspberry Pi に接続する時は以下のイメージのように接続します。

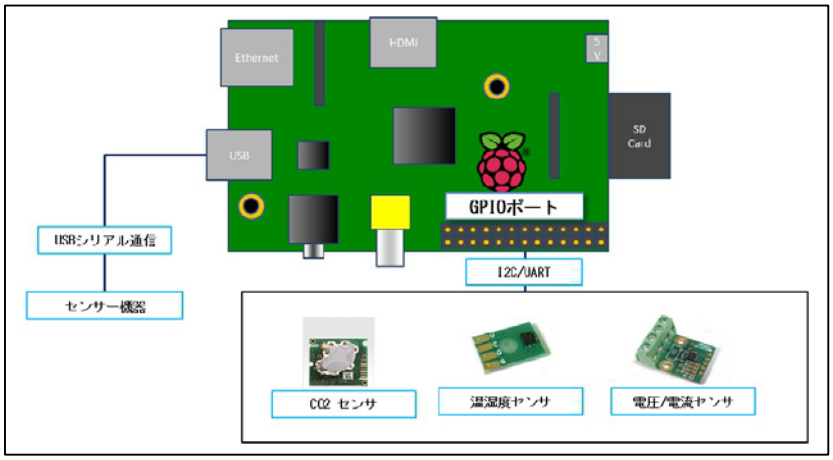


図 39：センサ接続イメージ

（※）GPIO接続のセンサーユニットは、5Vまたは3.3V電源、3.3V TTLレベルI/O対応のものであれば、直接接続可能です。

次にアクチュエータの接続方法を示します。アクチュエータで制御可能な機器は、リレーを１つ利用した[ON/OFF]によるスイッチ制御アクチュエータが最大8系統、リレーを２つ利用した[ON/OFF/正/逆]による[0%～100%]のポジション制御アクチュエータが最大4系統まで接続できます。

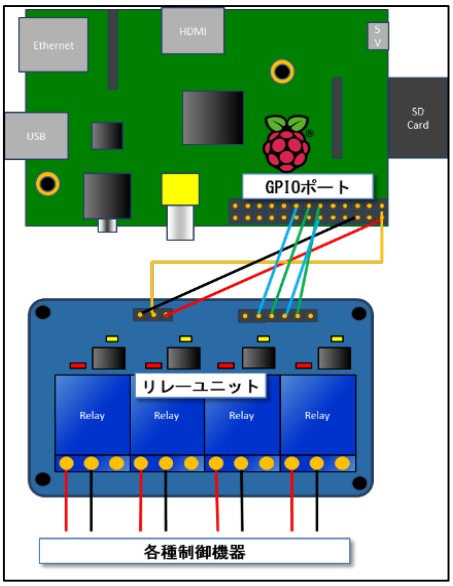


図 40：アクチュエータ接続イメージ

GPIO端子は以下のようにアサインされているので、利用目的に沿った形でセンサやリレーユニットと接続して下さい。

図 41：GPIO ピン番号

ピン番号	名称	機能説明	備考
1	3.3V DC Power	3.3V 電源	
2	5.0V DC power	5.0V 電源	
3	SDA0(I2C)	I2C シリアルデータ（SDA）	
4	5.0V DC Power	5.0V 電源	
5	SCL0(I2C)	I2C シリアルロック（SCL）	
6	GND	グランド	
7	GPIO 7	デジタル OUT [L /H]	
8	TxD	UART TX	
9	GND	グランド	
10	RxD	UART RX	
11	GPIO 0	デジタル OUT [L /H]	
12	GPIO 1	デジタル OUT [L /H]	
13	GPIO 2	デジタル OUT [L /H]	
14	GND	グランド	
15	GPIO 3	デジタル OUT [L /H]	
16	GPIO 4	デジタル OUT [L /H]	
17	3.3V DC Power	3.3V 電源	
18	GPIO 5	デジタル OUT [L /H]	

19	SPI MOSI	SPI 用ピン	
20	GND	グランド	
21	SPI MISO	SPI 用ピン	
22	GPIO 6	デジタル OUT [L /H]	
23	SPI SCLK	SPI 用ピン	
24	SPI CE0 N	SPI 用ピン	
25	GND	グランド	
26	SPI CE1 N	SPI 用ピン	

表 7：GPIO 端子アサイン表

3.5. WebUI 機能

UECS-Pi SDK の WebUI は、フレームワークに Apache Wicket（以降 Wicket）を利用しています。Wicket はオブジェクト指向型の Web アプリケーションフレームワークで、Web ページのデザインや、Web ページの部品をオブジェクトとして扱います。これらは独自拡張可能で、Wicket はこのようなオブジェクトを組み合わせて様々な Web ページを作成します。また UECS-Pi SDK では基本的な機能を持つページをあらかじめ準備しています。

3.5.1. 基本ページレイアウトとレイアウトの継承

WebUI 作成の際は、親ページクラスのレイアウトを継承して、子ページクラスを作成します。例えば UECS-Pi SDK のトップページ、CCM 一覧ページ等の全ての画面ページは、LayoutPage というクラスを継承する事で、そのレイアウトを継承しています。LayoutPage は、HeaderPanel、MenuPanel、CostomFeedbackPanel、FooterPanel といった部分表示クラスを内包しています。

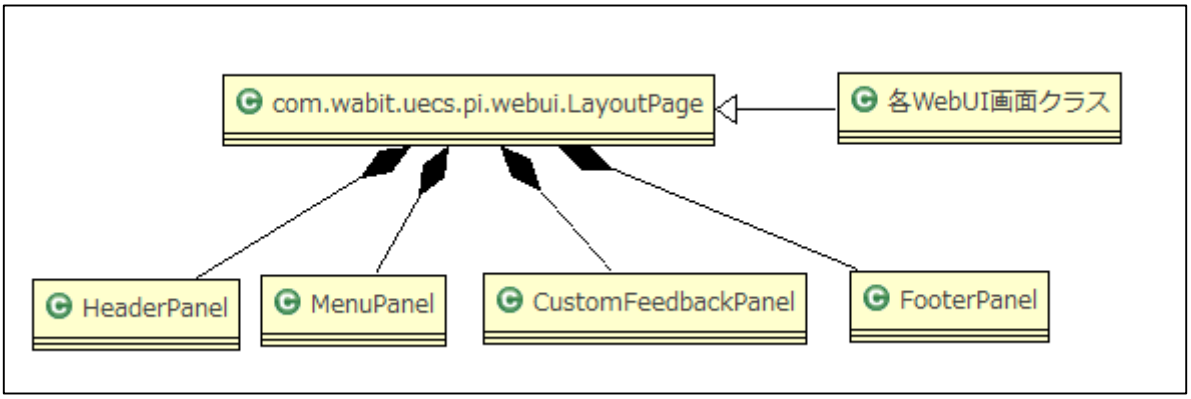


図 42：LayoutPage クラスと他の全ての WebUI クラスの関係

LayoutPage は、ベースのレイアウトクラスとして機能するために、LayoutPage.html という対になる HTML ファイルを持っています。これが継承されるレイアウトの大枠になります。同様に、内包する部分表示クラスにも対になる HTML ファイルが存在します。これらが合成された結果が最終的に HTML コードとして出力されます。

図 43：LayoutPage.java が形成しているレイアウト

上図の右側にあるのが、LayoutPage.html に集約されているページ部品です。これらの HTML ファイルに出力されるコンテンツや、ユーザ操作時の挙動は、対になる Java クラスに記載されています。

すなわち Wicket では、Java クラスはデータや動作を記述するファイルとして、HTML ファイルは Java クラスに記述されたデータや動作を出力するテンプレートファイルとして機能します。

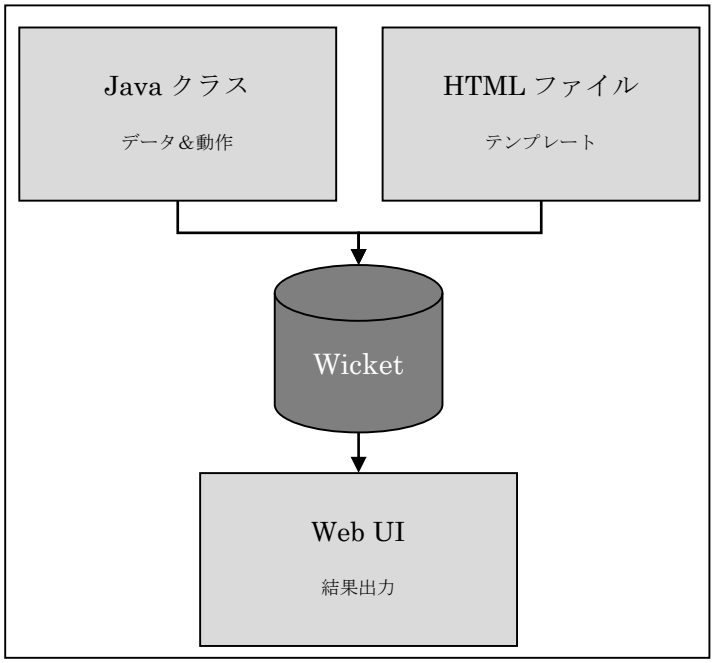


図 44：Wicket における Java ファイルと HTML ファイルの模式図

これら LayoutPage、ヘッダ、メニューパネル、コンテンツ、フッタの Java ファイルと HTML ファイルは、ページ部品のクラスライブラリとして機能します。

基本ページ一式の全てのページは、この LayoutPage.html のレイアウトを継承した上で、コンテンツ部分に各々のページ独自の内容を出力しています。これが Wicket の機能です。

開発者が接続デバイス用 WebUI を作成する際も、基本ページ一式と同じように LayoutPage.java を継承したクラスファイルを作り、このレイアウトを継承して作成していきます。継承したレイアウトをデザインのベースとして使う事で、開発者はコンテンツ部分に出力する接続デバイス用コンテンツ作成に注力出来ます。

またレイアウトを構成する Java クラスや HTML ファイルを独自に作成して差し替える事により、基本レイアウトを変更した WebUI を作成する事も可能です。（詳細はチュートリアルを参照）

3.5.2. WebUI 出力のためのファイルセット

Wicket では、基本的に Java ファイル（xxx.java）と HTML ファイル（xxx.html）が、1 つの Web ページを出力するためのファイルセットになります。また多言語表示対応させるためのリソースプロパティファイル（xxx.properties）をそこに加える事も出来ます。これらが連携して Web リクエストが来た時に表示される WebUI を作成します。

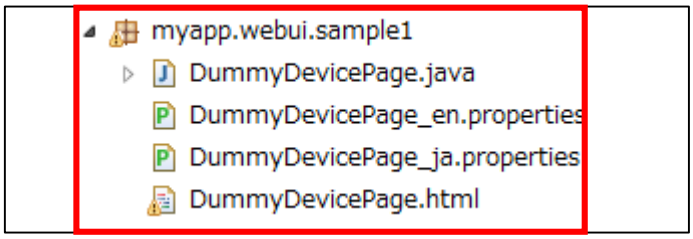


図 45：Web ページ出力のためのファイルセット例

各ファイルの機能概要は以下の通りです。

順序	動作名称	動作内容
1	Javaファイル	HTMLファイルに表示するデータや、HTTPリクエスト処理の挙動が記述されている
2	HTMLファイル	Javaファイルに記載した内容を反映するテンプレートになる。またプロパティファイルのテキストを表示するテンプレートにもなっている

3	プロパティファイル	各言語に対応したテキスト一式が記述されている。日本語で表示する時には日本語用（_ja）の、英語で表示する時には英語用（_en）のプロパティファイルのテキストが、HTMLファイルに表示される。これはブラウザのロケールによって切り替わる。多言語対応させる必要がない場合は、HTMLファイルに直接テキストを書いても良い。

表 8：WebUI を構成する各ファイルの機能概要

3.5.3. 基本 WebUI 一式

ここでは UECS-Pi SDK に始めから用意されている基本 WebUI 一式のコンテンツと使い方を解説します。基本 WebUI は以下の 5 機能です。また UECS-Pi SDK はこれらの WebUI が持つ機能に加え、ログアウト機能を持っています。

No.	名称	機能
1	認証ページ	認証を得ていないユーザが UECS-Pi にアクセスした時に表示されるページ
2	トップページ	ログイン直後、あるいは上部メニューの「トップ」をクリックすると表示されるページ
3	CCM 一覧ページ	CCMの状態一覧を確認できるページ
4	状態ログページ	ノードやデバイスの状態を確認できるページ
5	セットアップページ	上部メニューのセットアップにマウスを重ねた際に、プルダウンで出てくるメニューをクリックすると表示されるページ。初期はノード用のみ。開発者はここに接続デバイス用の WebUI を作成する

表 9：基本ページ機能概要

3.5.3.1. トップページ

ログイン直後、あるいは上部メニューの「トップ」をクリックすると表示されるページです。UECS-Pi SDK の動作状態や、接続されているセンサとアクチュエータの情報が一覧化されて表示されます。「更新」ボタンをクリックすると、表示されている値が更新されます。

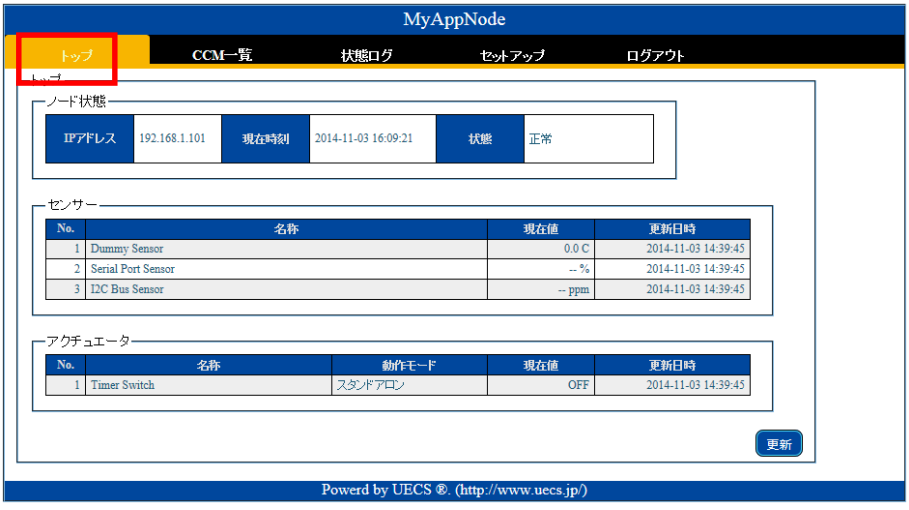


図 46：トップページ

3.5.3.2. 認証ページ

認証を得ていないユーザが UECS-Pi にアクセスした時に表示されるページです。

図 47：認証ページ

3.5.3.3. CCM 一覧ページ

上部メニューの「CCM 一覧」をクリックすると表示されるページです。UECS-Pi SDK に登録されている CCM(共用通信子)の状態が一覧化されて表示されます。「更新」ボタンをクリックすると、表示されている値が更新されます。

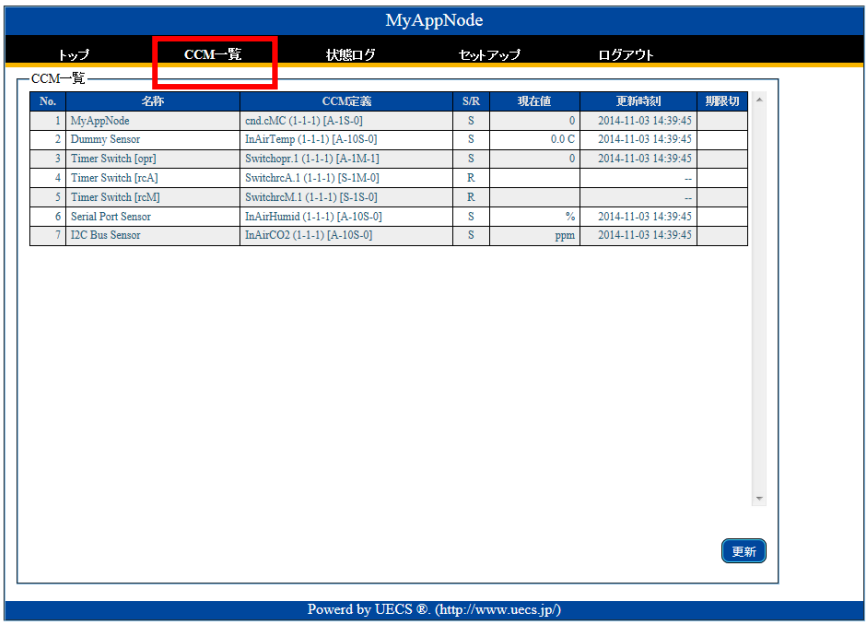


図 48：CCM 一覧ページ

3.5.3.4. 状態ログページ

上部メニューの「状態ログ」をクリックすると表示される画面です。各「カテゴリ」に分類されたログ情報が最新順に一覧化されて表示されます。「カテゴリ」には、ノード、デバイス、その他、全体があります。

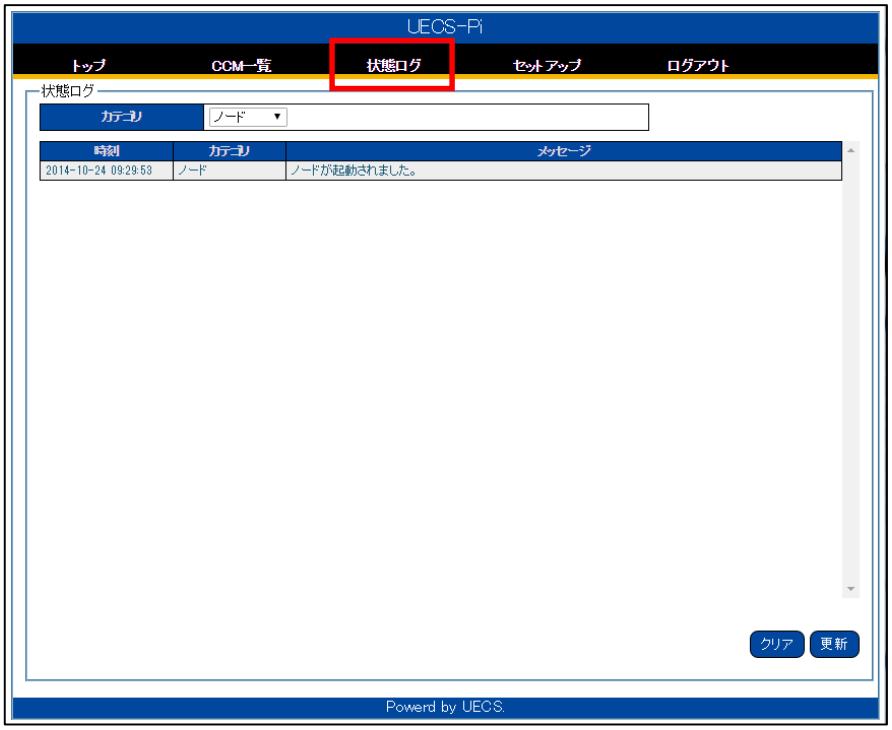


図 49：状態ログページ

ログカテゴリを変更した時に表示される内容を下表に示します。

No.	項目	説明
1	ノード	UECS-Pi 全体のノード機能に関するログ情報が出力されます。主に、起動・停止ログ、ネットワーク接続エラー情報などが含まれます。
2	デバイス	設定登録されたセンサやアクチュエータに関するエラー情報などが出力されます。
3	その他	その他付帯機能に関するログが出力されます。
4	全体	全てのカテゴリのログが一覧表示されます。

表 10：ログカテゴリ一覧

3.5.3.5. セットアップページ

上部メニューのセットアップにマウスを重ねた際に、プルダウンで出てくるメニューをクリックすると表示される画面です。ノードや各種デバイスの設定を行うページですが、はじめに用意されているのはノード設定ページのみなので、それについて説明します。ノード設定ページを表示するには、上部メニューから、「セットアップ」→「ノード設定」をクリックします。

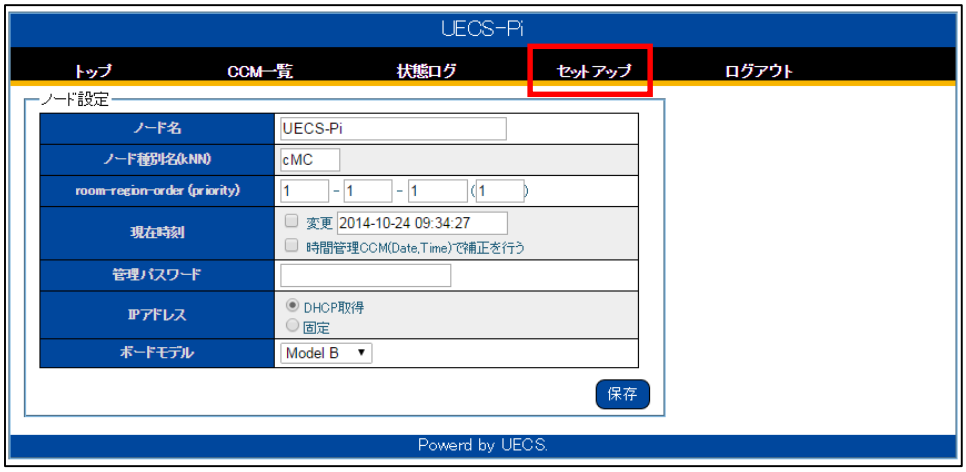


図 50：ノード設定ページ

ノード設定ページ使用時は、以下表を参考に接続動作させたい UECS/LAN ネットワークに合わせた設定値に変更し「保存」ボタンをクリックして設定を保存してください。

No.	項目	説明
1	ノード名	UECS 通信のノードスキャン応答 CCM の<NAME>項目に使用されます。設定名は画面上部のタイトルにも使用されます。設定文字は ASCII コード(ISO 646-1991)[半角の英数字記号]のみ使用が許可されています。
2	ノード種別(kNN)	UECS 通信で使用される CCM 識別子のノード種別に使用されます。ノード種別名の一覧は UECS 実用通信規約の仕様書に記載されています。
3	room-region-order (priority)	UECS 通信で使用される、各種区分番号です。LAN に接続される他の UECS ノードの設定値と重複しないように設定してください。 ・room ：部屋番号 [0～127] ・region ：系統番号 [0～127] ・order ：通し番号 [0～30000] ・priority：優先順位 [0～30]
4	現在時刻	「変更」をチェックして時刻を修正入力すると、Raspberry Pi 内部の時刻が変更されます。 「時間管理 CCM(Date, Time)で補正を行う」をチェックすると、UECS 通信

		規約で定義されている時間管理サーバが発信する CCM(Date, Time)を受信して、内部時刻を自動的に補正します。 （注:インターネット通信可能な LAN に接続されている場合は、自動的に NTP サーバから現在値が取得されます。NTP サーバに接続できず、Date/Time CCM も受信できない場合は、電源が OFF になると、現在時刻もリセットされます。）
5	管理パスワード	設定画面にログインするためのパスワードを変更したい場合に、新しいパスワードを入力します。
6	IPアドレス	Raspberry Pi 本体の IP アドレスを設定します。 ・「DHCP」LAN 内の DHCP サーバから自動的に IP アドレス情報を取得します。 ・「固定 IP」設定値項目が表示されますので、接続する LAN 環境に合わせた設定値を入力してください。 （注：UECS-Pi SDK のデフォルトでは「DHCP」になっていますが、Eclipse 上で開発している状態では IP アドレス変更は機能しません。Raspberry Pi 実機にインストールした際に、必要に応じて設定変更してください。）

表 11：ノード設定画面項目

4. チュートリアル

UECS-Pi SDK では様々なセンサを使った環境測定や、アクチュエータを使った機器制御が行えます。ここでは先の章で Eclipse にインポートした「UecsPi_SDK」プロジェクトファイル内に含まれているサンプルコードの解説をします。センサとアクチュエータを使う簡単なチュートリアルを紹介し、その基礎を理解します。作成したソフトウェアのインストール方法は「UECS-Pi アプリケーションのインストール」の章を参照して下さい。

4. 1. プロジェクト構成

Eclipse にインポートした「UecsPi_SDK」プロジェクトに含まれるフォルダやファイルについて概説します。

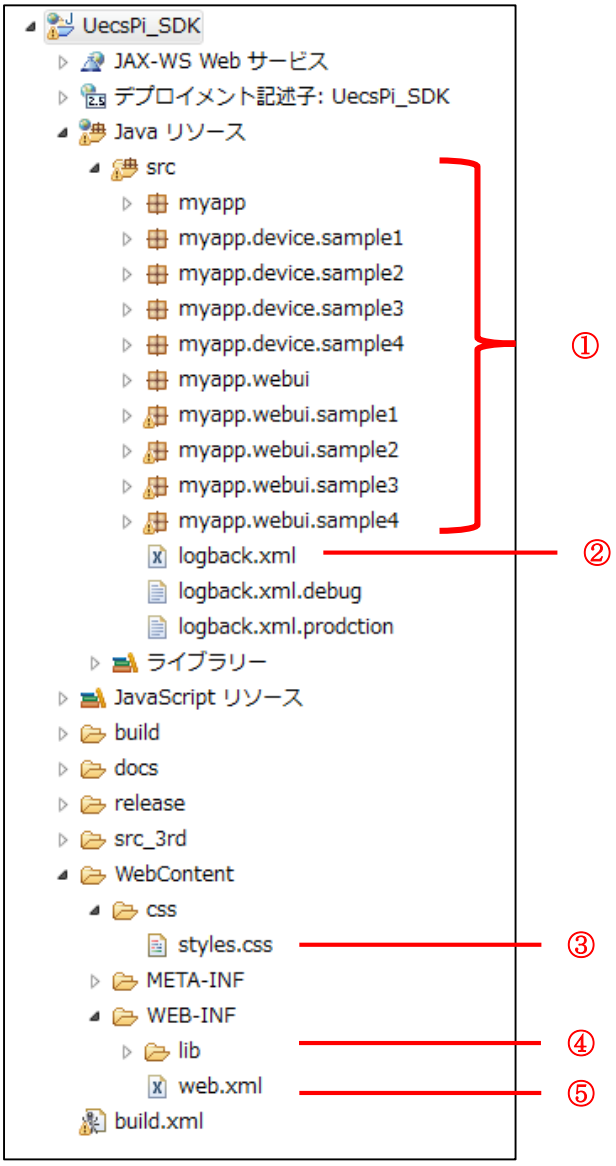


図 51：プロジェクト内フォルダ構成

No.	フォルダ/ファイル名	説明
1	/src/以下フォルダ	アプリケーションのソースコードがパッケージ構成ツリーの形でフォルダ分けされて格納されます。 「myapp」はサンプル用の仮パッケージ名ですので、任意のパッケージ名に変更可能です。
2	/src/logback.xml	プログラム内部出力ログの設定ファイルです。 内部デバッグ出力の切り替えなど、アプリケーション開発時と本番稼働時で設定を変更することで出力内容を切り替えることが可能です。設定ファイルの書式は、オープンソースログ出力ライブラリの Logback の書式に従います。 (http://logback.qos.ch/)
3	/WebContent/css/styles.css	WebUI 画面の配色や文字サイズなどを定義する CSS ファイルです。開発者は独自の CSS に差し替えることで画面デザインを変更可能です。
4	/WebContent/WEB-INF/lib	アプリケーションが動作時に参照するライブラリ jar ファイルが格納されています。開発者が独自の機能を開発するために必要なライブラリ jar を追加することが可能です。
5	/WebContent/WEB-INF/web.xml	Tomcat が読み込む Web アプリケーション設定ファイルです。Wicket 関連の設定が記述されています。

表 12：フォルダ、ファイル説明

4.2. ノード、WebUI アプリケーションクラス

まず始めに、UECS-Pi SDK で独自のアプリケーションを作成する際に基本となる UECS ノードクラスと WebUI アプリケーションクラスの解説をします。

4.2.1. ノードクラス

ノード部分の基本的な構成は「ノードクラス＋ノード設定クラス」が 1 セットとなります。

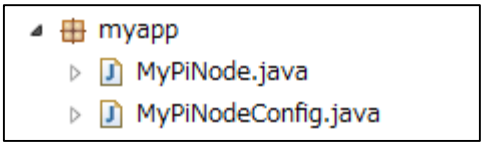


図 52：ノード部分のファイル構成

それでは、ノードクラスとノード設定クラスのコードを見ていきます。MyPiNode クラスは、UECS-Pi 基本ライブラリに用意されている親クラス(UecsPiNode)を継承することで UECS ノードとしての基本機能を持たせています。追加のコードとして、installProcess()メソッドをオーバーライドすることで、初回起動時にデバイスやコンポーネントを内部ＤＢに初期登録させています。本サンプルでは、チュートリアルで解説する４つのデバイスクラスを初期登録しています。動作させるデバイスが固定している場合は、初期登録しておくことが簡単ですが、初期起動時にはデバイスを登録せず、WebUI 画面でユーザが設定変更する操作と連動して、動的にデバイスやコンポーネントを登録、削除させることも可能です。

MyPiNode.java

```
public class MyPiNode extends UecsPiNode {

    public MyPiNode() {
        super(new MyPiNodeConfig());
    }

    protected void installProcess() throws Exception {

        DatabaseManager.callInTransaction(new Callable<Void>() {
            public Void call() throws Exception {

                // Sample 1
                DummyDevice dummyDevice = new DummyDevice(DatabaseUtils.nextDeviceId());
                DummySensor dummySensor = new DummySensor(DatabaseUtils.nextComponentId());
                dummyDevice.addComponent(dummySensor);
                DatabaseUtils.saveDevice(dummyDevice);

                // Sample 2
                SimpleGpioDevice gpioDevice = new SimpleGpioDevice(DatabaseUtils.nextDeviceId());
                TimerSwitchActuator timerSwitch = new
TimerSwitchActuator(DatabaseUtils.nextComponentId());
                gpioDevice.addComponent(timerSwitch);
                DatabaseUtils.saveDevice(gpioDevice);

                // Sample 3
                SimpleSerialDevice serialDevice = new SimpleSerialDevice(DatabaseUtils.nextDeviceId());
                RegexSensor regexSensor = new RegexSensor(DatabaseUtils.nextComponentId());
                serialDevice.addComponent(regexSensor);
                DatabaseUtils.saveDevice(serialDevice);

                // Sample 4
                SimpleI2cDevice i2cDevice = new SimpleI2cDevice(DatabaseUtils.nextDeviceId());
                I2cCo2Sensor co2Sensor = new I2cCo2Sensor(DatabaseUtils.nextComponentId());
                i2cDevice.addComponent(co2Sensor);
                DatabaseUtils.saveDevice(i2cDevice);

                return null;
            }

        });

    }

}
```

MyPiNodeConfig.java も基本ライブラリに用意されている親クラス(UecsPiNodeConfig)を継承することで、デフォルト設定値がプリセットされています。独自の設定値に変更したい場合に、コンストラクタ内で任意の初期設定値に書き換えることができます。本サンプルでは、ノード名称を独自のものに変更していますが、そのほかにも UECS ID やベンダー名、ノード状態通知 CCM の room-region-order なども初期値を変更できます。変更可能な設定値キーは、継承元クラスの定数で定義されていますので、JavaDoc を参照してください。

MyPiNodeConfig.java

```
public class MyPiNodeConfig extends UecsPiNodeConfig {

    public MyPiNodeConfig() {
        // ノード名称
        setString(KEY_NODE_NAME, "MyAppNode");
    }

}
```

4.2.1. WebUI アプリケーションクラス

WebUI アプリケーション部分では、Wicket フレームワークを利用した基底クラスを拡張し、画面パーツクラス（xxxPanel.java, xxx）、HTML テンプレートファイル（xxxPanel.html）、プロパティファイル（xxx.properties）を作成します。

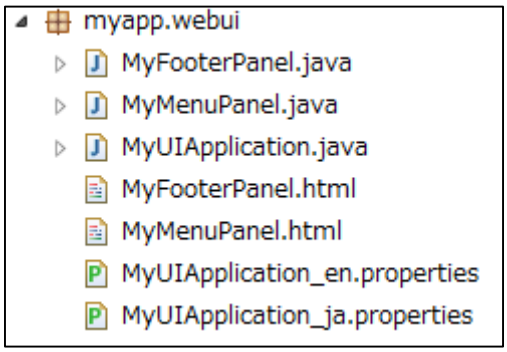


図 53：WebUI アプリケーション部分のファイル構成

本サンプルで作成している MyUIApplication クラスでは、先に作成した MyPiNode クラスをアプリケーション内で呼び出し、カスタマイズされたメニュー(MyMenuPanel)とフッター（MyFooterPanel）に変更するために、継承元クラスのメソッドをオーバーライドしています。

MyUIApplication.java

```
public class MyUIApplication extends WebUIApplication {
```

```
    @Override
    protected UecsPiNode createNodeInstance() {
        return new MyPiNode();
    }

    @Override
    protected Class<? extends Panel> getMenuPanelClass() {
        return MyMenuPanel.class;
    }

    @Override
    protected Class<? extends Panel> getFooterPanelClass() {
        return MyFooterPanel.class;
    }

}
```

各クラスの実装は Wicklet の API に従っていますので、詳細は Wicket の HP（http://wicket.apache.org/）のユーザーガイドを参照してください。

4.3. Sample 1 ： ダミーセンサ

このチュートリアルでは、ダミー値を送信する仮想センサのサンプルを解説します。実際のセンサ機器は使いません。WebUI から設定値をセンサ値としてノードに認識させ、定義変更可能な任意のデータ CCM として送信を行います。

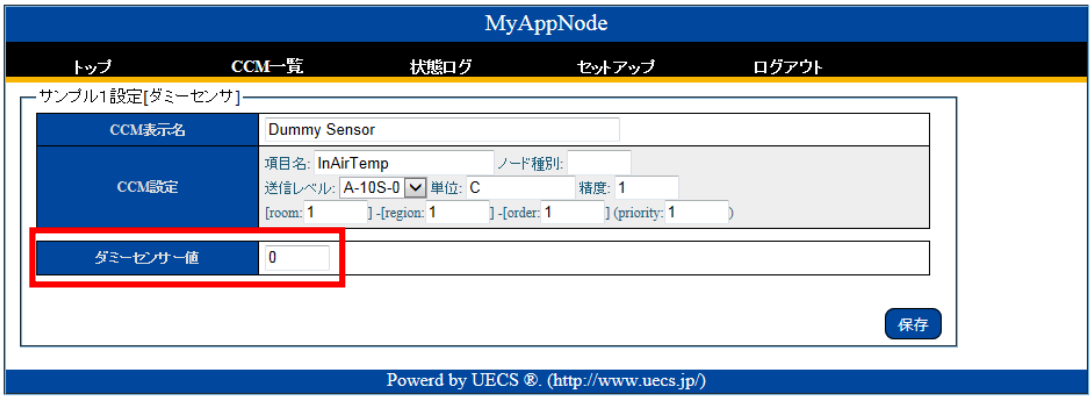


図 54：ダミーセンササンプル WebUI（枠内がダミーセンサ入力値）

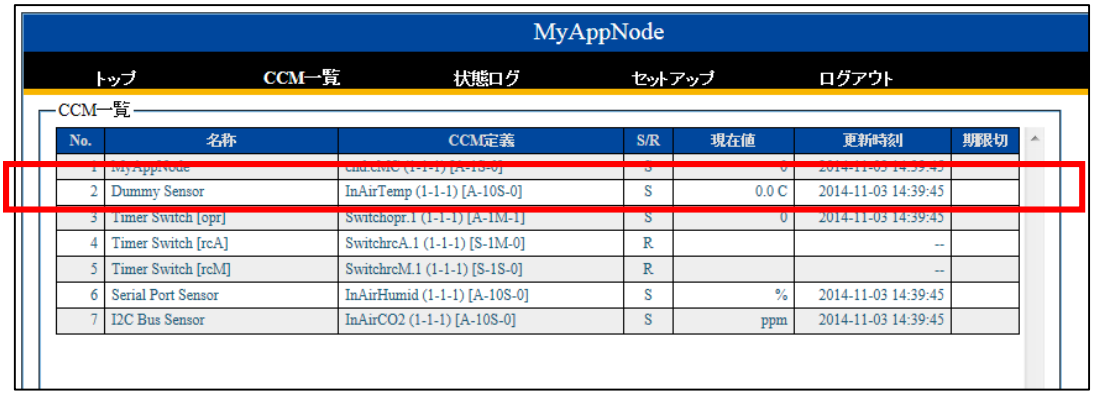


図 55：CCM 一覧画面（枠内がダミーセンサ入力値を反映した CCM）

4.3.1. デバイス、コンポーネントクラス

デバイス部分の基本的な構成は「デバイスクラス＋デバイス設定クラス」が 1 セットと、「コンポーネントクラス＋コンポーネント設定クラス」が任意セット数含まれる形です。本サンプルでは、コンポーネントはダミーセンサー１個となりますので、１セット用意されています。

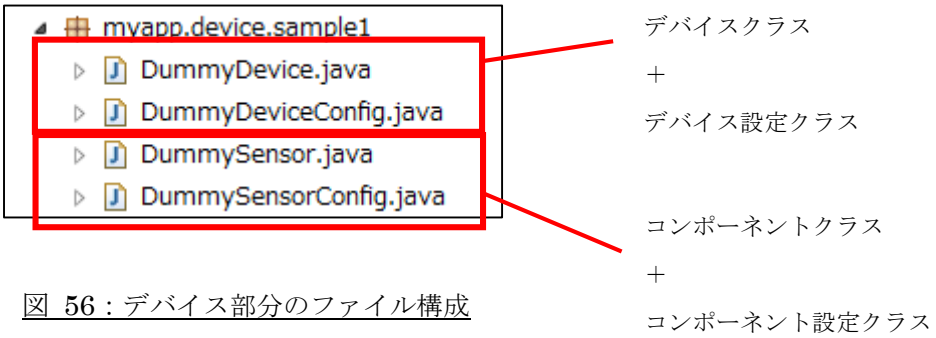


図 56：デバイス部分のファイル構成

それでは、デバイスクラスとデバイス設定クラスのコードを見ていきます。DammyDevice.java、DammyDeviceConfig.java は継承元の UECS-Pi 基本ライブラリクラスの PiDeviceBase、PiDeviceConfig と同じ動作をするコードになっていますので、ここでは説明は割愛します。コンポーネントクラスとコンポーネント設定クラスを見ていきます。

DammySensor.java

```
public class DummySensor extends PiSensorBase<DummySensorConfig> {

    public DummySensor(String id) {
        super(id, new DummySensorConfig());
    }

    @Override
    protected void onStart() throws Exception {
        // 起動時の処理を記述します
        Number val = getConfig().getFloat(DummySensorConfig.KEY_SENSE_VALUE, 0);
        super.setValue(val);
    }
```

```
    @Override
    protected void onStop() throws Exception {
        // 停止時の処理を記述します
    }


}
```

DammySensor.java も、基本的には継承元のクラス PiSensorBase と同じ動作をするコードですが、起動時に DammySensorConfig から KEY_SENSE_VALUE を取得して、setValue()に入力しています。この KEY_SENSE_VALUE が、WebUI から入力される、ダミーのセンサ取得値です。また setValue()は値が入力されると、センサーコンポーネントに対応する CCM の値も自動的に更新されます。

DammySensorConfig.java

```
public class DummySensorConfig extends PiSensorConfig {

    /** 設定値項目キー：ダミーセンサー値 */
    public final static String KEY_SENSE_VALUE = "SenseValue";

    public DummySensorConfig() {
        // センサー値に対応するCCM情報を初期設定します。
        setString(KEY_COMPONENT_NAME, "Dummy Sensor");
        setString(KEY_CCM_INFO_NAME, "InAirTemp");
        setString(KEY_CCM_LEVEL, CcmLevel.A_1OS_0.toString());
        setInt(KEY_CCM_ROOM, 1);
        setInt(KEY_CCM_REGION, 1);
        setInt(KEY_CCM_ORDER, 1);
        setInt(KEY_CCM_PRIORITY, 1);
        setString(KEY_CCM_UNIT, "C");
        setString(KEY_CCM_SIDE, "S");
        setInt(KEY_CCM_CAST, 1);
        setInt(KEY_SENSE_VALUE, 0);
    }

}
```

DammySensorConfig.java では、コンポーネント設定値の定数定義と対応する CCM の初期設定を行っています。これらの初期値は、ダミーセンサ用の WebUI の初期値として使用されます。コンポーネントに設定できる値のキー定数は、継承元クラスに定義されていますので、JavaDoc を参照してください。

4.3.1. WebUI 設定画面クラス

次は、デバイス設定画面の WebUI コード部分を見ていきます。WebUI 部分の基本的な構成は「Java ファイル＋HTML ファイル＋プロパティファイル」です。

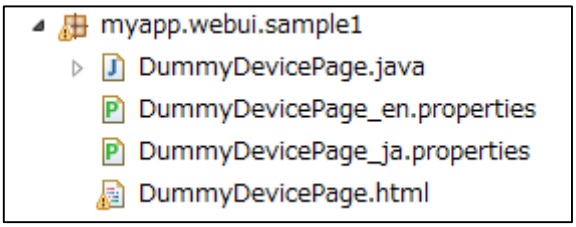


図 57：WebUI 部分のファイル構成

DammyDevicePage.java

```
@AuthorizeInstantiation({Roles.ADMIN}) ------①
public class DummyDevicePage extends LayoutPage {  ------②
    private static final long serialVersionUID = 9205561784306869639L;

    private Form<PropertyConfig> deviceForm;
    private PropertyConfig formParams = new PropertyConfig(); ------③

    private Log logger = LogFactory.getLog(getClass());

    private static String[] ccmLevels = new String[] {
            CcmLevel.A_1S_0.toString(),
            CcmLevel.A_1S_1.toString(),
            CcmLevel.A_10S_0.toString(),
            CcmLevel.A_10S_1.toString(),
            CcmLevel.A_1M_0.toString(),
            CcmLevel.A_1M_1.toString(),
            CcmLevel.B_0.toString(),
            CcmLevel.B_1.toString(),
    };

    private String deviceId;
    private String componentId;

    public DummyDevicePage() {
    }

     .
     .<中略>
     .

    @Override
    protected void onInitialize() {
        super.onInitialize();

        // 設定対象となるデバイスとコンポーネントのIDを取得します。
        MyPiNode node = (MyPiNode)MyUIApplication.getNodeInstance();
        DummyDevice device = node.listDevices(DummyDevice.class).get(0); ------④
        DummySensor sensor = device.listComponents(DummySensor.class).get(0); ------⑤
        deviceId = device.getId();
        componentId = sensor.getId();

        // コンポーネント設定値をWebUIフォームの初期値としてコピーします
```

```
        formParams.putAll(sensor.getConfig());

        deviceForm = new Form<PropertyConfig>("deviceForm") {
            private static final long serialVersionUID = 1L;

            @Override
            public void onSubmit() {    ------⑥
                try {
                    deviceForm.modelChanging();

                    // フォーム入力された設定値をセンサー設定に反映して
                    // DBに保存したのち、ノードを再起動します。
                    DatabaseManager.callInTransaction(new Callable<Void>() {
                        public Void call() throws Exception {
                            DummySensor sensor = (DummySensor)MyUIApplication
                                    .getNodeInstance().getDevice(deviceId)
                                    .getComponent(componentId);
                            sensor.getConfig().putAll(formParams);
                            DatabaseUtils.saveComponent(deviceId, sensor);
                            MyUIApplication.getNodeInstance().restart();
                            return null;
                        }
                    });

                    deviceForm.modelChanged();
                    success(MessageUtils.getMessage(MessageCode.SAVED));
                } catch (Exception e) {
                    error(MessageUtils.getMessage(MessageCode.ERROR));
                    logger.error("save error.", e);
                }
            }
        };

        // フォームに入力フィールドを登録します。    ------⑦
        deviceForm.setDefaultModel(new CompoundPropertyModel<PropertyConfig>(formParams));
        deviceForm
                .add(new RequiredTextField<String>(DummySensorConfig.KEY_COMPONENT_NAME, String.class)
                        .add(new StringValidator(1, 50)));
        deviceForm.add(new RequiredTextField<String>(DummySensorConfig.KEY_CCM_INFO_NAME,
String.class)

         .
         .<中略>
         .

        add(deviceForm);

    }


}
```

DammyDevicePage.java では HTML ファイルに埋め込み表示するコンテンツや、HTML が操作された時の処理を記述しています。コンテンツを作成しているコードでは、主に Wicket フレームワークが用意している Form クラスに対して、入力フィールドのアイテムや、入力値バリデーションクラスの登録を記述しています。

No.	コードの説明
1	Webページ表示の際に、認証チェックを行うためのアノテーション
2	LayoutPageクラスを継承する事で、UECS-Pi SDKのWebUIデザインを継承
3	Web画面の入力フォームフィールドの入力値を格納するためのデータオブジェクト

4	ノードに初期登録されているデバイスインスタンスを取得
5	デバイスに初期登録されているセンサーコンポーネントインスタンスを取得
6	Web画面の登録ボタンクリックのリクエスト処理を記述
7	WicketのFormオブジェクトに入力フィールドデータを関連付け、各項目に対応するフィールドオブジェクトや、入力バリデーションオブジェクトを生成して登録

表 13：DammyDevicePage.java の処理

次は、HTML ファイルを見ていきます。「3.4.1.基本ページレイアウトとレイアウトの継承」で説明したように、WebUI の HTML ファイルは、Java ファイルに記述されている内容を表示するテンプレートになっています。

DammyDevicePage.html

```
・
・<中略>
・
<body>
   <wicket:extend> ------①
      <fieldset style="width: 780px;">
         <legend>
            <wicket:message key="title.dummy" /> ------②
         </legend>
         <form wicket:id="deviceForm"> ------③
            <table border="1" style="width: 770px">
               <tr>
                  <th style="width: 180px;"><wicket:message
                        key="ComponentName" /></th>
                  <td><input type="text"
                     wicket:id="ComponentName" size="50" ------④
                     style="width: 300px" /></td>
               </tr>
                ・
                ・<中略>
                ・

            </table>
            <table border="1" style="width: 770px">
               <tr>
                  <th style="width: 180px;"><wicket:message key="SenseValue" /></th>
                  <td><input type="text" wicket:id="SenseValue" size="5" style="width: 50px" /></td>
               </tr>
            </table>
            <br />
            <div style="text-align: right;">
               <input type="submit" value="Save"  ------④
                  wicket:message="value:button.save" />
            </div>
         </form>
      </fieldset>
   </wicket:extend>
</body>
</html>
```

DammyDevicePage.html では、LayoutPage を継承していますが、コンテンツは独自のものを表示します。独自コンテンツが表示されるのは<wicket:extend>～</wicket:extend>タグ内です。DammyDevicePage.java の deviceForm の内容は<form wicket:id=”deviceForm”>～</form”>に表示されています。

また DammyDevicePage.java のデータ入力オブジェクトと HTML ファイル内のフォーム入力フィールド値を「wicket:id」属性値で互いに関連付けています。

No.	コードの説明
1	独自コンテンツ表示用タグ
2	wicket:messageタグ。プロパティファイルからテキストを取得している
3	DammyDevicePage.javaのdeviceFormの内容を反映するタグ
4	wicket:idでJavaクラス内のformParamsオブジェクトの値と関連付けている
5	deviceFormAreaにaddされた内容を呼び出している
6	「保存」ボタン。クリックするとJavaクラス内のFormオブジェクトのsubmitメソッド処理が行われる

表 14：DammyDevicePage.html のタグの意味

4.4. Sample 2 ： タイマー動作アクチュエータ

このチュートリアルでは、Raspberry Pi の GPIO 端子にリレーユニットを接続し、一定間隔で周期的に ON/OFF 動作をするサンプルを解説します。実際に機器同士を接続して動作を確認して下さい。

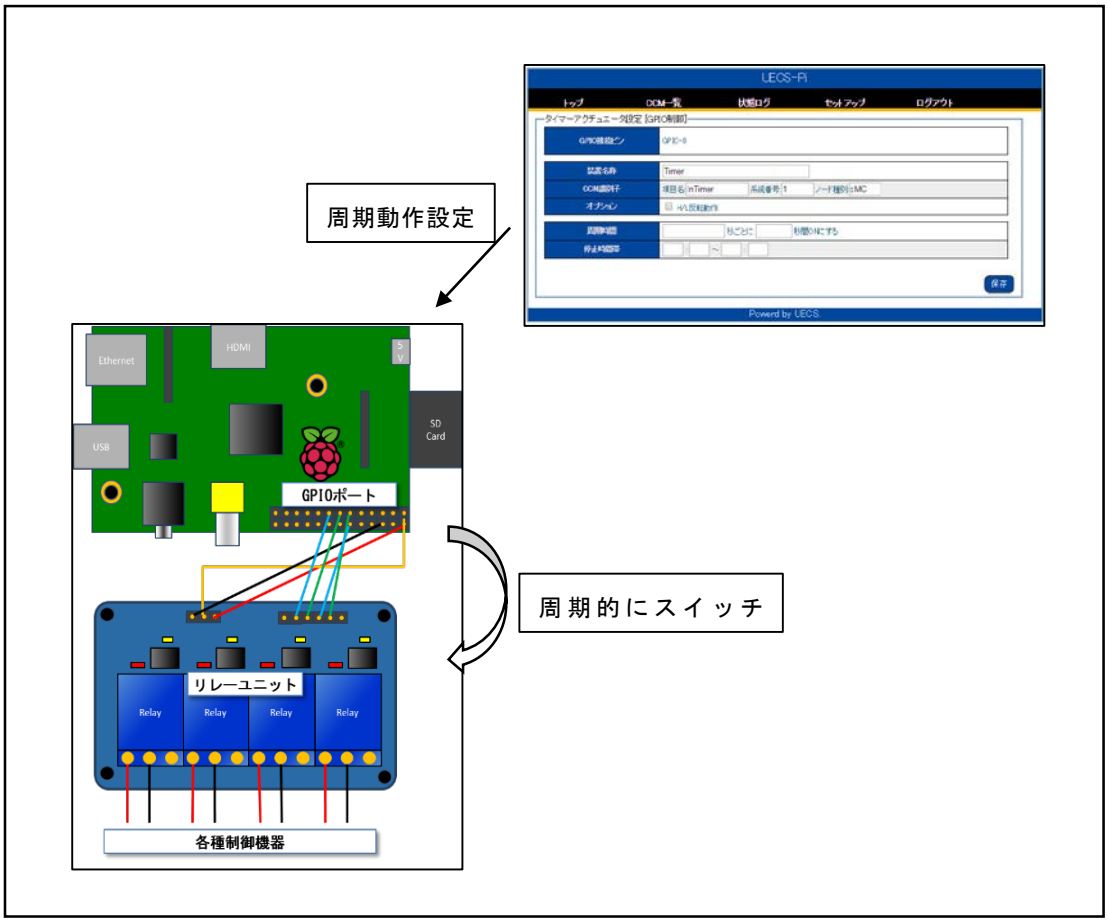


図 58：タイマー動作アクチュエータサンプル作成イメージ

WebUI 画面から、各種動作定義を行うことができます。

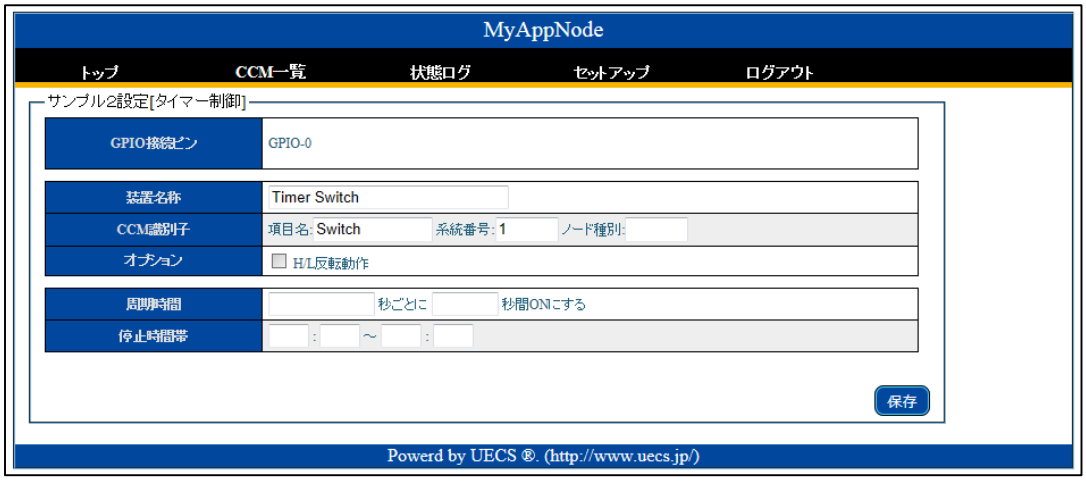


図 59：タイマーアクチュエータサンプル WebUI 設定項目

No.	項目	説明
1	装置名称	接続した機器のラベル名称です。トップ画面などでアクチュエータ情報として表示される名称となります。
2	CCM識別子	UECS 通信で使用される CCM 識別子の情報です。設定可能な識別子のルールは、UECS 実用通信規約、あるいは運用ガイドライン書類に記載されていますので、参考に設定してください。 [設定例]項目名=Relay, 系統番号=1, ノード識別子=cMC で設定された場合は、 以下の 3 種類の CCM がノード内部に登録されます。 ・制御機器運転状態 CCM (送信)=Relayopr.1.cMC ・遠隔制御指示 CCM (受信)=RelayrcA.1.cMC ・遠隔操作指示 CCM (受信)=RelayrcM.1.cMC
3	オプション	[H/L 反転動作] デフォルト動作では GPIO ピン状態が、LOW=OFF、HIGH=ON として扱われます。 接続しているリレーモジュールによっては動作が逆の場合がありますので、その場合はチェックを入れてください。
4	周期時間	周期条件を入力します。 例：”[30]秒ごとに[10]秒間 ON にする”と入力された場合、 [ON:10 秒]-[OFF:20 秒]-[ON-10 秒]-[OFF:20 秒]～（以降繰り返し）～
5	停止時間帯	一日のうち、定期動作を停止したい時間帯がある場合に指定できます。 0 時をまたいだ設定も可能です。（例：23:00～2:00）

表 15：タイマーアクチュエータサンプル WebUI 設定項目解説

4.4.1. デバイス、コンポーネントクラス

デバイス部分のコードを解説します。

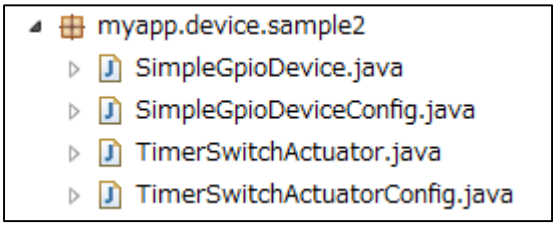


図 60：デバイス部分のファイル構成

SimpleGpioDevice.java

```
public class SimpleGpioDevice extends GpioDeviceBase<SimpleGpioDeviceConfig> {

    public SimpleGpioDevice(String id) {
        super(id, new SimpleGpioDeviceConfig());
    }

    @Override
    protected void onInit() throws Exception {
        super.onInit();

        UecsPiNodeConfig nconf = (UecsPiNodeConfig) getNode().getConfig();
        for (IComponent<?> compo : listComponents()) {
            // アクチュエータCCMの属性値として、ノードのCCM属性値を引き継いで設定
            ComponentConfig conf = compo.getConfig();
            conf.setInt(ComponentConfig.KEY_CCM_ROOM, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ROOM,
0));
            conf.setInt(ComponentConfig.KEY_CCM_REGION,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_REGION, 0));
            conf.setInt(ComponentConfig.KEY_CCM_ORDER, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ORDER,
0));
            conf.setInt(ComponentConfig.KEY_CCM_PRIORITY,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_PRIORITY, 0));
        }
    }

}
```

SimpkeGpioDeviceConfig.java

```
public class SimpleGpioDeviceConfig extends GpioDeviceConfig {

    private static final long serialVersionUID = -2238752941334340868L;

    public SimpleGpioDeviceConfig() {

    }

}
```

デバイスクラスでは内包するコンポーネント(ここではTimerSwitchActuator)のCCMの属性値(room / region / order / priority)をノードの設定値と同期させるために引継ぎコピーを行っています。デバイス設定クラスは継承元のクラスと同じ動作を行っています。

次はアクチュエータクラス、アクチュエータ設定クラスを見ていきます。

TimerSwitchActuator.java

```
public class TimerSwitchActuator extends SwitchActuator<TimerSwitchActuatorConfig> {

    public TimerSwitchActuator(String id) {
        super(id, new TimerSwitchActuatorConfig());

    }

    @Override
    protected void onInit() throws Exception {
        super.onInit();

        TimerSwitchActuatorConfig config = getConfig();
        int periodicInterval = config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_INTERVAL, 0);
        int onInterval = config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_ON, 0);

        if (periodicInterval > 0 && onInterval > 0) {
            // 設定値が有効な場合は、定周期動作
            PeriodicSwitchAction periodAction = new PeriodicSwitchAction();
            periodAction.setPeriodicInterval(periodicInterval * 1000L);
            periodAction.setOnInterval(onInterval * 1000L);

periodAction.setInactiveStartTime(config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_INACTIVE_START_HOUR, -1),
                    config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_INACTIVE_START_MIN, -1));

periodAction.setInactiveEndTime(config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_INACTIVE_END_HOUR, -1),
                    config.getInt(TimerSwitchActuatorConfig.KEY_PERIODIC_INACTIVE_END_MIN, -1));
            setAction(ActionMode.Autonomy, periodAction);
            setActionMode(ActionMode.Autonomy);
        } else {
            // 何も設定されない場合は、スタンドアローンモードで起動
            setActionMode(ActionMode.Standalone);
        }
    }


}
```

アクチュエータクラスでは初期化を行う onInit()メソッドをオーバーライドして、周期動作の設定を行っています。周期 ON/OFF 動作を行うアクションクラスは、UECS 基本ライブラリで提供されています(PeriodicSwitchAction)ので、そのクラスを利用するだけで、アクチュエータの動作を定義する事が可能です。ここでは、E10 規約で定義されている、アクチュエータの動作モードのうち、自律モード(ActionMode.Autonomy)に対応したアクションとして登録しています。定期動作以外にも、センサーCCM 連動や、時刻範囲指定で動作を行うようなアクションクラスも用意されていますので、詳細は JavaDoc を参照してください。

TimerSwitchActuatorConfig.java

```
public class TimerSwitchActuatorConfig extends SwitchActuatorConfig {

    private static final long serialVersionUID = 1L;

    /** 設定項目キー ： 定周期動作間隔 */
    public static final String KEY_PERIODIC_INTERVAL = "PeriodicInterval";
```

```
    /** 設定項目キー : ON動作間隔 */
    public static final String KEY_PERIODIC_ON = "PeriodicOn";
    /** 設定項目キー : 動作停止開始時刻(時) */
    public static final String KEY_PERIODIC_INACTIVE_START_HOUR = "PeriodicInactiveStartHour";
    /** 設定項目キー : 動作停止開始時刻(分) */
    public static final String KEY_PERIODIC_INACTIVE_START_MIN = "PeriodicInactiveStartMinute";
    /** 設定項目キー : 動作停止終了時刻(時) */
    public static final String KEY_PERIODIC_INACTIVE_END_HOUR = "PeriodicInactiveEndHour";
    /** 設定項目キー : 動作停止終了時刻(分) */
    public static final String KEY_PERIODIC_INACTIVE_END_MIN = "PeriodicInactiveEndMinute";

    public TimerSwitchActuatorConfig() {
        setInt(KEY_DEFAULT_VALUE, 0);
        setInt(KEY_FIXED_VALUE, 0);
        setBoolean(KEY_INVERSE, false);
        setBoolean(KEY_IS_REPRESENTATIVE, false);
        setInt(KEY_CCM_LINE, 1);
        setInt(KEY_GPIO_PIN, 0);
        setString(KEY_COMPONENT_NAME, "Timer Switch");
        setString(KEY_CCM_INFO_NAME, "Switch");
    }
}
```

TimerSwitchActuatorConfig.java では設定値キー定義と初期値設定を行っています。この定数は、WebUI 設定画面の入力フィールドのキーとしても使用されます。

4.4.2. WebUI 設定画面クラス

WebUI 部分のファイルの構成は Sample 1 と全く同じ構造です。実装パターンも同様ですので、ここでは説明を割愛します。

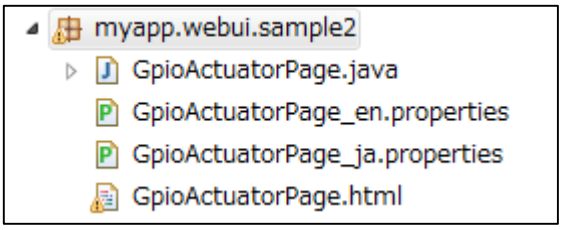


図 61：WebUI 部分のファイル構成

4.5. Sample 3：シリアル通信

個のチュートリアルでは、シリアル通信を使って外部接続されたデバイスから定期的に送信されるデータ文字列を読み取ってセンサ値として CCM 送信するサンプルを解説します。データ文字列は、正規表現パターンマッチングで数値部分を抽出します。

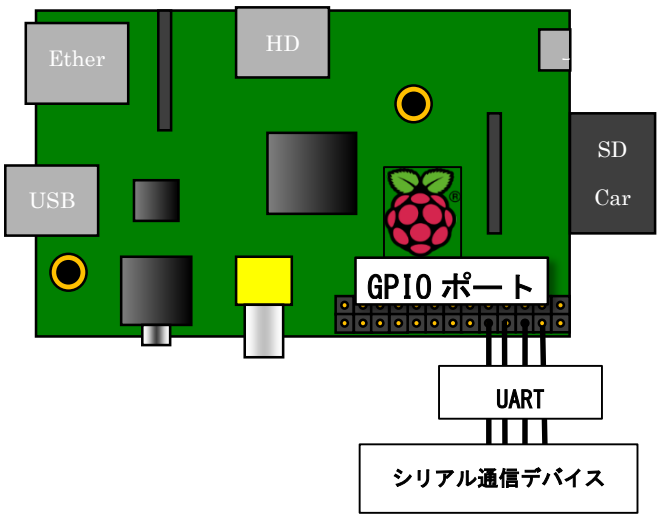
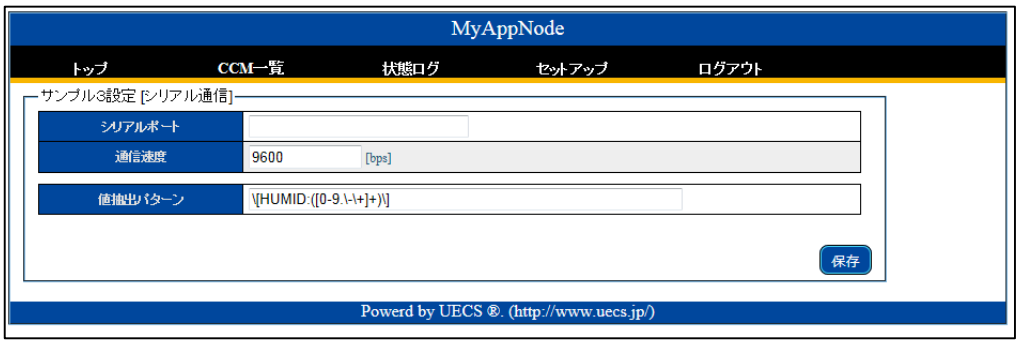


図 62：実際のシリアル通信デバイスとの接続イメージ



No.	項目	説明
1	シリアルポート	シリアルポートの識別子を入力します。通常は以下で動作します。 ・GPIO ポートの UART(RX/TX)接続：「/dev/ttyS0」 ・USB シリアル変換接続：「/dev/ttyUSB0」
2	通信速度	シリアルポートの通信速度(bps)を入力します。接続する機器側の通信速度の仕様に合わせてください。
3	値抽出パターン	値を抽出する正規表現パターンを設定します。(書式は Java 正規表現ライブラリに従います)

表 16：シリアル通信サンプル WebUI 設定項目解説

4.5.1. デバイス、コンポーネントクラス

デバイス部分のコードを解説します。

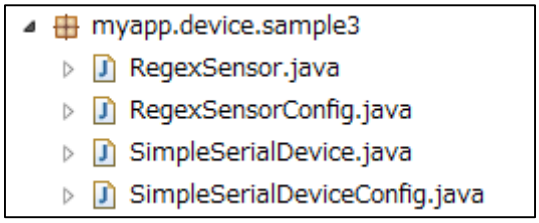


図 63：デバイス部分のファイル構成

SimpleSerialDevice.java

```
public class SimpleSerialDevice extends SerialPortDeviceBase<SimpleSerialDeviceConfig> {

    public SimpleSerialDevice(String deviceId) {
        super(deviceId, new SimpleSerialDeviceConfig());
    }

    @Override
    protected void onInit() throws Exception {
        super.onInit();

        UecsPiNodeConfig nconf = (UecsPiNodeConfig) getNode().getConfig();
        for (IComponent<?> compo : listComponents()) {
            // CCMの属性値として、ノードのCCM属性値を引き継いで設定
            ComponentConfig conf = compo.getConfig();
            conf.setInt(ComponentConfig.KEY_CCM_ROOM, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ROOM,
0));
            conf.setInt(ComponentConfig.KEY_CCM_REGION,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_REGION, 0));
            conf.setInt(ComponentConfig.KEY_CCM_ORDER, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ORDER,
0));
            conf.setInt(ComponentConfig.KEY_CCM_PRIORITY,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_PRIORITY, 0));
        }
    }

    @Override
    public void dataReceived(byte[] data) {
        for (RegexSensor sensor : listComponents(RegexSensor.class)) {
            sensor.parseData(data);
        }
    }


}
```

SimpleSerialDeviceConfig.java

```
public class SimpleSerialDeviceConfig extends SerialPortDeviceConfig {
    private static final long serialVersionUID = -5081030325821182630L;

    public SimpleSerialDeviceConfig() {
        setString(KEY_DEVICE_NAME, SimpleSerialDevice.class.getSimpleName());
        setString(KEY_SERIAL_PORT_ID, "");
        setInt(KEY_SERIAL_PORT_SPEED, 9600);
    }
```

```
}
```

UECS-Pi 基本ライブラリでは、シリアル通信を行う基底デバイスクラスが用意されています(SerialPortDeviceBase)ので、継承することでシリアル通信機能を持たせることができます。SimpleSerialDeviceConfig で設定された通信ポート ID(*KEY_SERIAL_PORT_ID*) と通信スピード[BPS](*KEY_SERIAL_PORT_SPEED*)で通信を行います。データが受信されると、非同期に dataReceived()メソッドが起動されますので、dataReceived()メソッドをオーバーライドして受信データを処理するコード記述します。本サンプルでは実デバイスに対してデータ送信は行っていませんが、継承元クラスには、データ送信を行う sendData()メソッドも用意されています。詳細は JavaDoc を参照してください。

RegexSensor.java

```
public class RegexSensor extends PiSensorBase<RegexSensorConfig> {
    private Pattern regexPattern;

    public RegexSensor(String id) {
        super(id, new RegexSensorConfig());
    }

    @Override
    protected void onInit() throws Exception {
        super.onInit();
        RegexSensorConfig conf = getConfig();
        String textPattern = conf.getString(RegexSensorConfig.KEY_REGEX_PATTERN);
        if (textPattern != null) {
            try {
                regexPattern = Pattern.compile(textPattern);
            } catch (PatternSyntaxException e) {
                notifyException(e);
            }
        }
    }

    @Override
    protected void onStart() throws Exception {
    }

    @Override
    protected void onStop() throws Exception {
    }

    public void parseData(byte[] data) {
        String text = null;
        try {
            text = new String(data, "us-ascii");
        } catch (UnsupportedEncodingException e) {
            return;
        }

        if (regexPattern != null && text != null) {
            Matcher matcher = regexPattern.matcher(text);
            if (matcher.find()) {
                String sval = matcher.group(1);
                try {
                    double value = Double.valueOf(sval);
                    setValue(value);
                } catch (NumberFormatException e) {
                    notifyException(e);
                }
            }
```

```
        }
    }
}
```

RegexSensorConfig.java

```
public class RegexSensorConfig extends PiSensorConfig {

    private static final long serialVersionUID = -2298819350796006077L;
    public static final String KEY_REGEX_PATTERN = "RegexPattern";

    public RegexSensorConfig() {
        setString(KEY_COMPONENT_NAME, "Serial Port Sensor");
        setString(KEY_CCM_INFO_NAME, "InAirHumid");
        setString(KEY_CCM_LEVEL, CcmLevel.A_IOS_0.toString());
        setString(KEY_CCM_UNIT, "%");
        setString(KEY_REGEX_PATTERN, "¥¥[HUMID:([0-9.¥¥-¥¥+]+)¥¥]");
    }

}
```

RegexSensor.java は WebUI 画面から設定された正規表現パターンを元に、シリアル通信で得られたバイト配列データを文字列に変換して値抽出を行っています。本サンプルでは、SerialDevice に対して 1 つの RegexSensor インスタンスを持たせていますが、複数の RegexSensor を持たせるように拡張すれば、異なる文字パターンを解析して複数のセンサー値を取得する機能も簡単に実現することができます。送受信データは byte 配列で処理されますので、バイナリ値での通信にも対応可能です。

4.5.2. WebUI 設定画面クラス

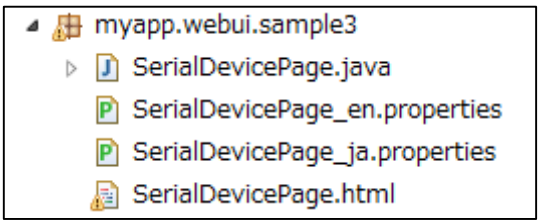


図 64：WebUI 部分のファイル構成

SerialDevicvePage.java は、Sample1、Sample2 と同じ構造です。また対になる HTML ファイル等も同様ですので、説明は割愛します。

4.6. Sample 4 ： I2C 通信

最後に、I2C 通信方式で仮想の CO2 センサデバイスと通信するサンプルを作成します。実在の製品デバイスとの通信ではありませんが、本サンプルコードを参考に、データ取得コマンドや受信データチェック処理などをコーディングする事で、実在の I2C 通信デバイスと通信が可能になります。

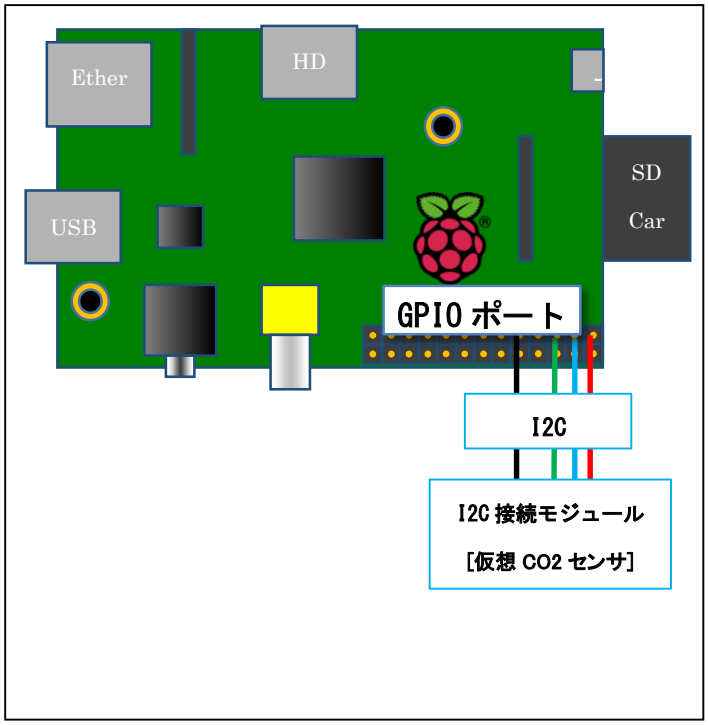
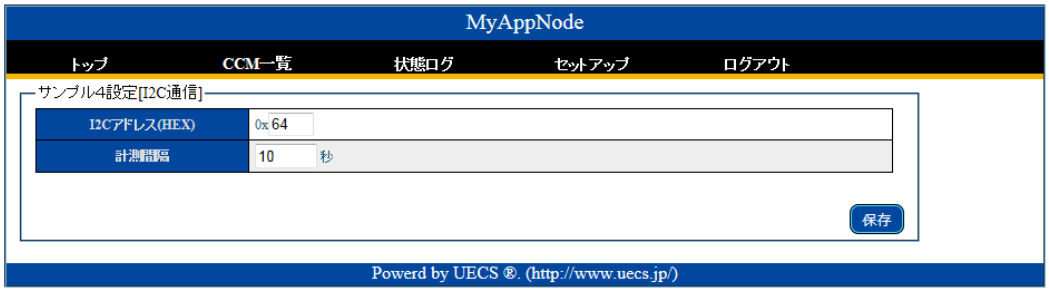


図 65：I2C 通信デバイス接続イメージ



No.	項目	説明
1	I2Cアドレス	I2C バスに接続されたデバイスのアドレス(HEX 形式)を設定します
2	計測間隔	定期的にセンサー値取得コマンドを発行する時間間隔を設定します

表 17：I2C 通信サンプル WebUI 設定項目解説

4.6.1. デバイス、コンポーネントクラス

デバイス部分のコードを解説します。

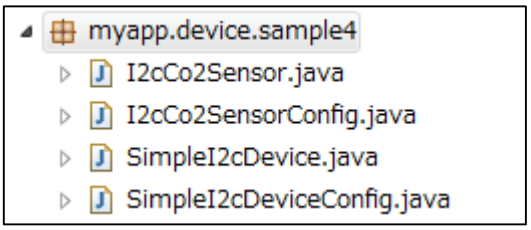


図 66：デバイス部分のファイル構成

SimpleI2cDevice.java

```
public class SimpleI2cDevice extends I2cDeviceBase<SimpleI2cDeviceConfig> {

    public SimpleI2cDevice(String deviceId) {
        super(deviceId, new SimpleI2cDeviceConfig());
    }

    @Override
    protected void onInit() throws Exception {
        super.onInit();

        UecsPiNodeConfig nconf = (UecsPiNodeConfig) getNode().getConfig();
        for (IComponent<?> compo : listComponents()) {
            // アクチュエータCCMの属性値として、ノードのCCM属性値を引き継いで設定
            ComponentConfig conf = compo.getConfig();
            conf.setInt(ComponentConfig.KEY_CCM_ROOM, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ROOM,
0));
            conf.setInt(ComponentConfig.KEY_CCM_REGION,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_REGION, 0));
            conf.setInt(ComponentConfig.KEY_CCM_ORDER, nconf.getInt(UecsPiNodeConfig.KEY_NODE_ORDER,
0));
            conf.setInt(ComponentConfig.KEY_CCM_PRIORITY,
nconf.getInt(UecsPiNodeConfig.KEY_NODE_PRIORITY, 0));
        }
    }
}
```

SimpleI2cDeviceConfig.java

```
public class SimpleI2cDeviceConfig extends I2cDeviceConfig {

    public SimpleI2cDeviceConfig() {
        setString(KEY_DEVICE_NAME, SimpleI2cDevice.class.getSimpleName());
        setInt(KEY_I2C_ADDRESS, 0x40);
    }

}
```

UECS-Pi 基本ライブラリでは、I2C 通信を行う基底デバイスクラスが用意されています(I2cDeviceBase)ので、継承することで I2C 通信機能を持たせることができます。SimpleI2cDeviceConfig で設定された I2C アドレス（*KEY_I2C_ADDRESS*）に対して通信を行います。

本サンプルでは１つだけの I2C アドレスに対して通信を行っていますが、ノードに対して複数のデバイスを持たせることで、複数の I2C アドレスデバイスと通信可能です。

I2cCo2Sensor.java

```
public class I2cCo2Sensor extends I2cSensorBase<I2cCo2SensorConfig> {

    private static final byte GET_CO2_DATA = (byte) 0x05;

    private byte[] buffer = new byte[2];

    public I2cCo2Sensor(String id) {
        super(id, new I2cCo2SensorConfig());
    }

    @Override
    public void onReadProcess(I2CDevice i2cDevice) throws Exception {

        I2cUtils.write(i2cDevice, 2, GET_CO2_DATA);
        int len = I2cUtils.read(i2cDevice, 100L, buffer, 0, 2);
        if (len >= 2) {
            int co2 = (buffer[0] & 0xff) * 256 + (buffer[1] & 0xff);
            setValue(co2);
        }

    }

}
```

I2cCo2SensorConfig.java

```
public class I2cCo2SensorConfig extends I2cSensorConfig {

    private static final long serialVersionUID = -3216421789130713701L;

    public I2cCo2SensorConfig() {
        setString(KEY_COMPONENT_NAME, "I2C Bus Sensor");
        setString(KEY_CCM_INFO_NAME, "InAirCO2");
        setString(KEY_CCM_LEVEL, CcmLevel.A_10S_0.toString());
        setString(KEY_CCM_UNIT, "ppm");
        setInt(KEY_CCM_CAST, 0);
        setInt(KEY_SENSE_INTERVAL, 10);
    }

}
```

I2cCo2Sensor では、周期動作をしてセンサ値を取得しています。ただし、その処理は継承元のクラス I2cSensorBase クラス内で自動的に行われるため、このクラスでは、周期動作タイミングで起動される、onReadProcess()メソッドをオーバーライドして受信処理を記述しています。周期動作間隔は、I2cSensorConfig で定義されている設定値(KEY_SENSE_INTERVAL)で変更可能です。

I2C センサクラスとの通信をする基本的な流れは、「センサクラス起動時に周期実行処理セット」→「処理タイミングが来たら onReadProcess 起動」→「I2cUtil.write()で、データ取得コマンド発行」→「I2cUtil.read()で、データを読み取り、値を setValue()で登録」→「CCM の値が更新される」という流れになります。

4.6.2. WebUI 設定画面クラス

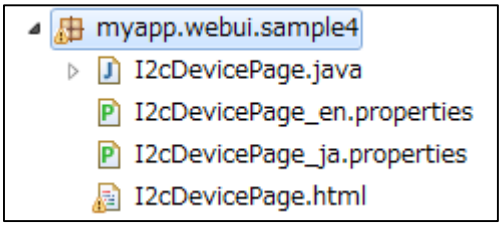


図 67：WebUI 部分のファイル構成

I2cDevicvePage.java は、Sample1、Sample2、Sample3 と同じ構造です。また対になる HTML ファイル等も同様ですので、説明は割愛します。

5. UECS-Pi アプリケーションのインストール

ここでは Raspberry Pi の初期設定、及び UECS-Pi SDK で開発したアプリケーションを Raspberry Pi にインストールする手順について説明します。

5.1. Raspberry Pi セットアップ

No.	名称	説明	関連情報
1	Raspberry Pi Model B/B+	UECS-Pi SDK動作製品本体	販売サイトURL http://jp.rs-online.com/web/
2	ACアダプタ	Raspberry Pi の microUSB 電源端子に 5V-750mA 以上を給電可能なもの（周辺機器を接続する場合 1.0～1.2A 程度を推奨）	
3	SDカード＋SDカードリーダ	SDHC 4GB（Class4）以上 ※Type B+の場合は microSD	
4	HDMI接続ディスプレイ+USBキーボード	Raspberry Pi の初期設定時に接続して使用	
5	LAN接続ハブ/ケーブル	Raspberry Pi と 開発用 PC をイーサネット経由で接続するための機器とケーブル	
6	Raspbian OS	Raspberry Pi 上で動作する Linux OS	ダウンロードURL http://www.raspberrypi.org/downloads/
7	Win32 Disk Imager	RAW ディスクイメージを SD カードに書き込むソフトウェア ※Mac OS/Linux を使われる方は、Win32 Disk Imager ではなく「dd」コマンドで書き込みが可能です。	ダウンロードURL http://sourceforge.jp/projects/sfnet_win32diskimager/releases/

表 18：Raspberry Pi セットアップに必要な機器とソフトウェア

5. 1. 1. SD カードの準備

① Raspbian OS の配布ページに移動し、zip ファイルをダウンロードします。

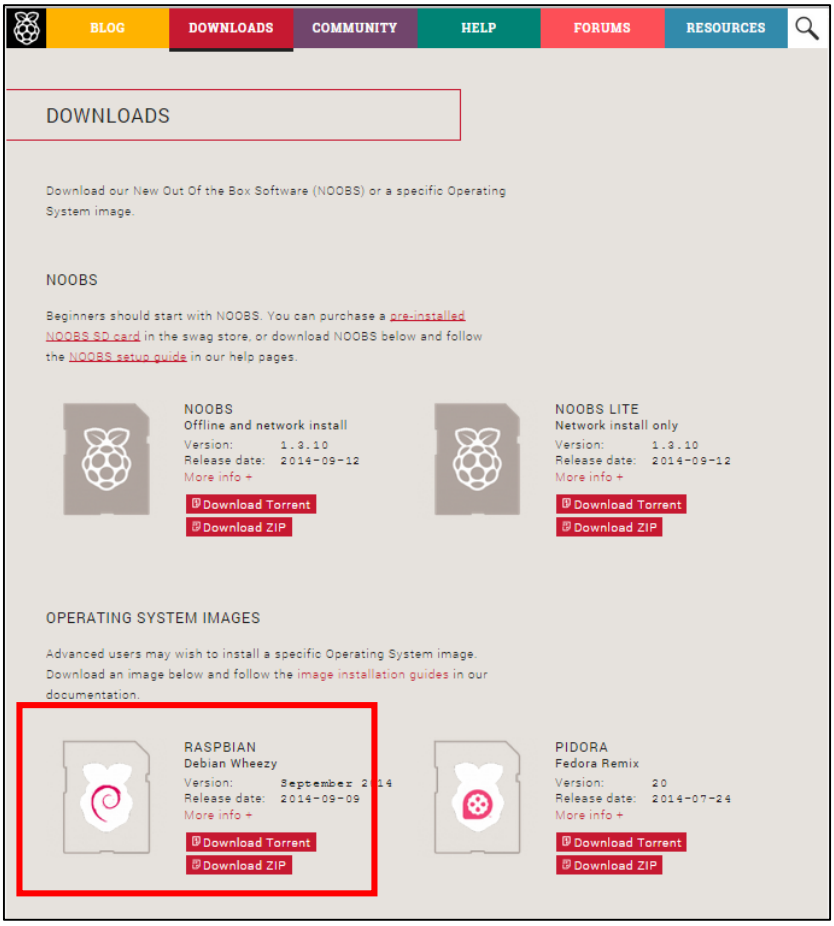


図 68：Rapbian OS 配布ページ

② ダウンロードが完了したら、ファイルを適当な場所に解凍します。

③ Win32 Disk Imager 配布サイトに移動し、exe ファイルをダウンロードしてインストールします。

④ 使用する SD カードを SD カードスロットに差し込み、その後 Win32 Disk Imager を起動します。

⑤ フォルダのアイコンをクリックして、OS のイメージファイル（例：2014-06-20-wheezy-raspbian.img）を選択します。「Write」ボタンをクリックして、SD カードへの書き込みを行います。

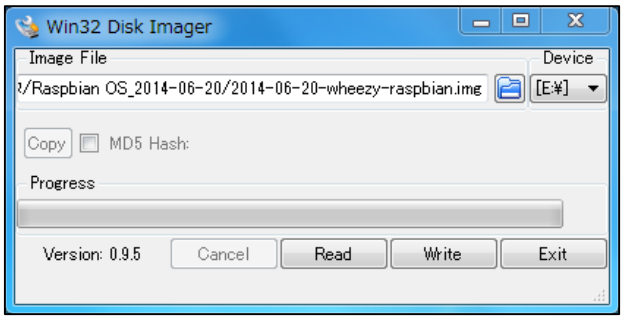


図 69：Win32 Disk Imager 画面

⑥ 書き込み終了後、SD カードを SD カードスロットから取り外し、Win32 Disk Imager を終了してください。

5.1.2. Raspberry Pi の初期設定

① Raspberry Pi の SD カードスロットに OS イメージファイルを書き込んだ SD カードを差し込みます。次にキーボード、ディスプレイ、LAN ケーブル等の周辺機器を Raspberry Pi に接続します。最後に AC アダプタを接続すると Raspberry Pi が起動します。

② 起動後、下記のような Raspberry Pi Software Configuration Tool(raspi-config)画面が表示されます。

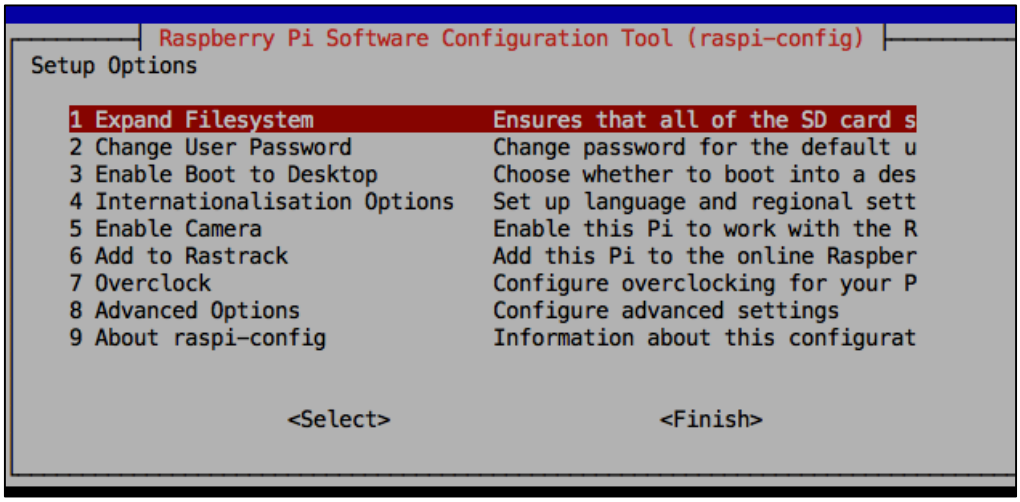


図 70：Raspberry Pi Software Configuration Tool(raspi-config)画面

以下、順に設定を行ってください。

A) 1 Expand Filesystem

Enter を押すと次の画面が表示されます。「OK」を選択してください

B) 2 Change User Password

Enter を押すと、画面下部で新しいパスワードを尋ねられますので、2 回、同じパスワードを入力してください。

※以降、Raspberry Pi にログインを行う際に必要になりますので忘れないようにしてください

C) 3 Enable Boot to Desktop/Scratch

設定する必要はありません。

D) 4 Internationalisation Options

Enter を押すと下記のような言語、タイムゾーン、キーボードレイアウトのメニュー画面に

遷移します。

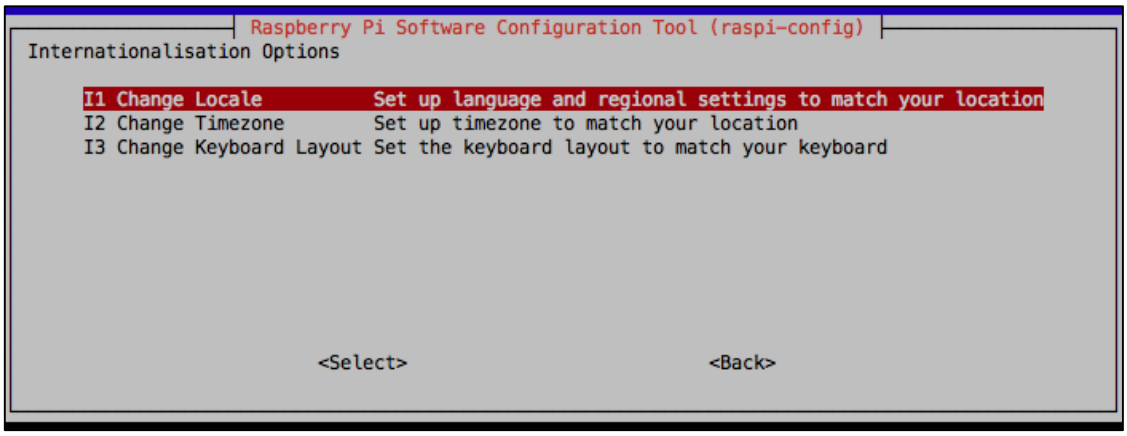


図 71：言語、タイムゾーン、キーボードレイアウトのメニュー画面

以下、順に設定を行ってください。

No.	メニュー	説明
1	I1 Change Locale	en_US.UTF-8、ja_JP.UTF-8 の二つを有効にします。有効にするにはスペースキーを押して下さい。デフォルトロケールは en_US.UTF-8 に設定してください。
2	I2 Change Timezone	Asia を選択した後、Tokyo を選択します。
3	I3 Change Keyboard Layout	下記の順に選択してください ①Generic 105-key (Intl) PC ②Other ③Japanese ※Japanese (PC-98xx Series)ではないので注意してください ④Japanese - Japanese(OADG 109A) ⑤The default for the keyboard layout ⑥No Compose Key ⑦Yes

表 19：Internationalisation Options の設定

E) 5 Enable Camera

設定する必要はありません。

F) 6 Add to Rastrack

設定する必要はありません。

G） 7 Overclock

設定する必要はありません。

H） 8 Advanced Options

Enter を押すと下記のような詳細メニュー画面に遷移します。

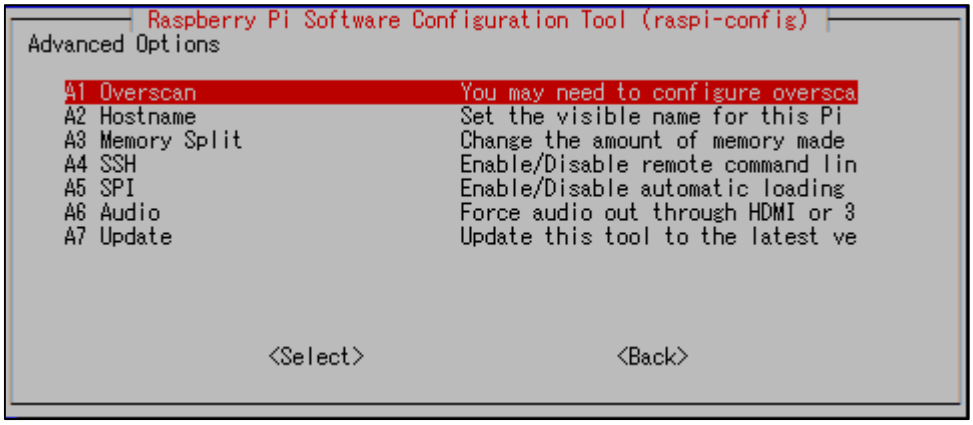


図 72：詳細メニュー画面

以下、順に設定を行ってください。

No.	メニュー	説明
1	A1 Overscan	設定する必要はありません。
2	A2 Hostname	設定する必要はありません。
3	A3 Memory Split	16（MB）と入力して OK を選択してください。
4	A4 SSH	Enable を選択してください。
5	A5 SPI	実行する必要はありません。
6	A6 Audio	実行する必要はありません。
7	A7 Update	実行する必要はありません。

表 20：Advanced Options の設定

上記の設定を完了後、Finish を選択してください。再起動の可否を尋ねられますので Yes を選択して、Raspberry Pi の再起動を行ってください。

5.1.3. Raspberry Pi のユーザ設定

① Raspberry Pi の再起動後、下記のようにログインユーザを尋ねられます。

```
raspberrypi login: _
```

② 下記のログインユーザとパスワードでログインしてください。
User：pi
Password：『5.1.2. Raspberry Pi の初期設定』で設定したパスワード
（変更していない場合、パスワードは”raspberry”になっています）

③ ログインが成功した場合、画面には下記のように“ユーザ@ raspberrypi”と表示されます。

```
pi@raspberrypi ~ $
```

④ 次に root ユーザのパスワードを変更します。下記の通りに入力してください。

```
pi@raspberrypi ~ $ sudo passwd root
```

⑤ パスワードを尋ねられますので、2 回、同じパスワードを入力してください。
※このパスワードは root ユーザのパスワードになります。初期設定で設定したもの（pi ユーザ）とは違いますので注意してください。

⑥ パスワード変更に成功後、実行ユーザを root ユーザに切り替えます。

```
pi@raspberrypi ~ $ su
```

⑦ パスワードを尋ねられますので、先ほど設定した root ユーザのパスワードを入力してください。
ユーザが切り替わると下記のように画面表示が変更されます。

```
root@raspberrypi:/home/pi#
```

以降の手順は全て、この root ユーザで実行してください。

5. 1. 4. IP アドレス設定

① 下記の通りに入力し、ネットワークの設定ファイルを開いてください。

```
root@raspberrypi:~# vi /etc/network/interfaces
```

② 本設定ファイルでは IP アドレス、サブネットマスク、ゲートウェイアドレスの設定を行います。
下記の通りにファイルの内容を変更してください。

```
auto lo

iface lo inet loopback
#iface eth0 inet dhcp

#allow-hotplug wlan0
#iface wlan0 inet manual
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf

auto eth0
iface eth0 inet static
address 192.168.0.xxx (←マシンの固定 IP アドレス)
netmask xxx.xxx.xxx.xxx (←サブネットマスク)
gateway xxx.xxx.xxx.xxx (←ゲートウェイアドレス)
```

5.1.5. DNS 設定

① 下記の通りに入力し、ネットワークの設定ファイルを開いてください。

```
root@raspberrypi:~# vi /etc/resolv.conf
```

② 本設定ファイルでは DNS の設定を行います。下記の通りにファイルの内容を変更してください。

```
nameserver xxx.xxx.xxx.xxx (←DNS のアドレス)
```

③ 設定が完了しましたら、Raspberry Pi の再起動を行ってください。

```
root@raspberrypi:~# reboot
```

5.1.6. シリアル通信(UART)の有効化

① Raspberry Pi のシリアル通信用ポート(RX/TX ピン)を利用可能に設定します。
"/boot/cmdline.txt "を編集します。

```
root@raspberrypi:~# vi /boot/cmdline.txt
```

② 下記の通りにファイルの内容を変更してください。(改行しているように見えますが、実際は一行としてください)

```
dwc_otg.lpm_enable=0 rpitestmode=1 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline rootwait
```

③ 次に、"/etc/inittab "を編集します。

```
root@raspberrypi:~# vi /etc/inittab
```

④ 以下の行(最終行)をコメントアウトしてください。

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

⑤ 次に、" /etc/rc.local"を編集します。

```
root@raspberrypi:~# vi /etc/rc.local
```

⑥ 下記の通りにファイルの内容を変更してください。

```
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s¥n" "$_IP"
fi

ln -s /dev/ttyAMA0 /dev/ttyS0
exit 0
```

5.1.7. I2C ポートの有効化

① 初期設定では Raspberry Pi 基盤上の I2C/SPI ポートは無効になっているため有効化します。
"/etc/modprobe.d/raspi-blacklist.conf "を編集します。

```
root@raspberrypi:~# vi /etc/modprobe.d/raspi-blacklist.conf
```

② 下記の通りにファイルの内容を変更して下さい。

```
# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

③ 認識させます。

```
root@raspberrypi:~# modprobe i2c-dev
```

④ 起動時に自動でロードさせるため、"/etc/modules "に i2c-dev を追加します。

```
root@raspberrypi:~# vi /etc/modules
```

```
# etc/modules: kernel modules to load boot time.
…
…

snd-bcm2835
i2c-dev
```

5.1.8. Tomcat の設定

No.	名称	説明	ダウンロード URL
1	Tomcat	オープンソースのHTTPサーバ	http://tomcat.apache.org/download-70.cgi
2	WinSCP	MS-Windows上で動く フリーのFTP、FTPS、SFTPクライアントプログラム。	http://winscp.net/eng/download.php
2	PuTTY	フリーの Telnet/SSH クライアント。SSH は SSH1 と SSH2 プロトコルが実装されている。	http://hp.vector.co.jp/authors/VA024651/PuTTYkj.html

表 21：Tomcat の設定に必要なソフトウェア

アプリを動作させるために、まず Tomcat（Web サーバ）のインストールと設定を行います。ここでは Tomcat7 を対象に説明を行います。またここまでで Raspberry Pi のネットワーク設定が完了しているので、ここからは PuTTY と WinSCP を使った操作を行います。

① Raspberry Pi からディスプレイ、キーボード等を取り外し、SD カード、ネットワークケーブル、AC アダプタのみが接続されている状態にします。

② WinSCP（または他の SCP クライアントソフト）をダウンロードして起動します。ここでは WinSCP を例に説明を行います。

③ WinSCP のログイン画面から「新しいサイト」をクリックし、下図にあるようにデータを入力します。ユーザ名やホスト名は、これまで設定した内容を入力して下さい。ここではルートユーザでログインし、ルートフォルダで作業する事を想定します。

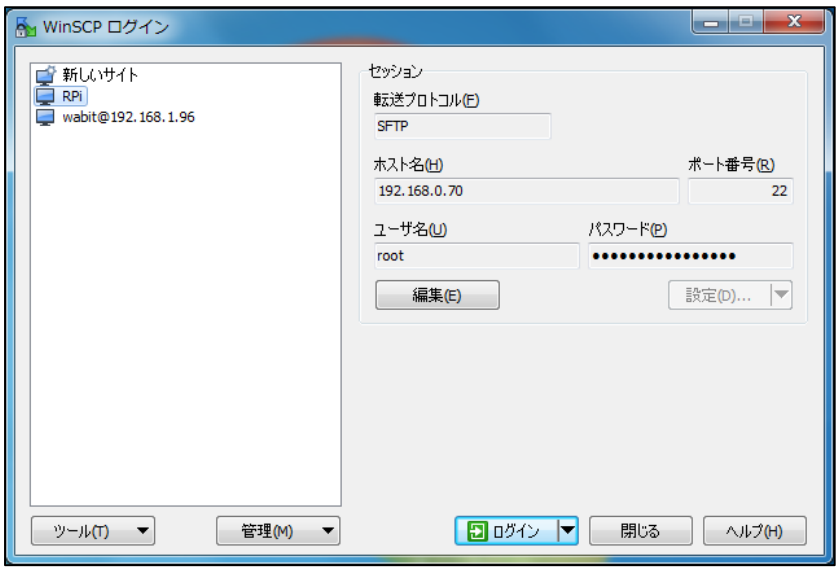


図 73：WinSCP ログイン画面

④　「ログイン」ボタンをクリックし、パスワードを入力して Rapberry Pi にログイン出来れば成功です。

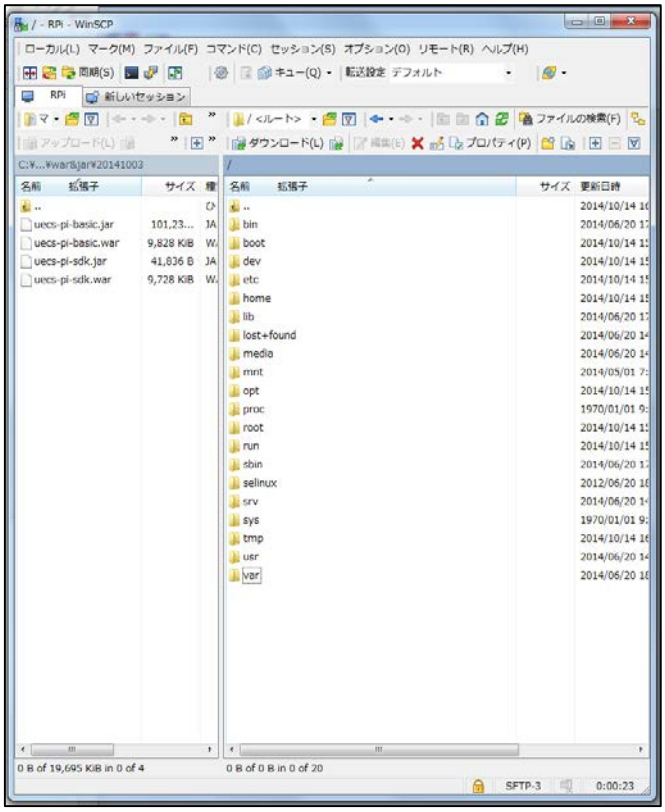


図 74：WinSCP ログイン後画面（画像の画面はルートフォルダ）

⑤ Tomcat をインストールするために、ブラウザ等で公式サイトのダウンロードページ（http://tomcat.apache.org/download-70.cgi）を開き、Core という項目の tar.gz リンクをクリックしてファイルをダウンロードします。バージョンは最新のものをダウンロードしてください。

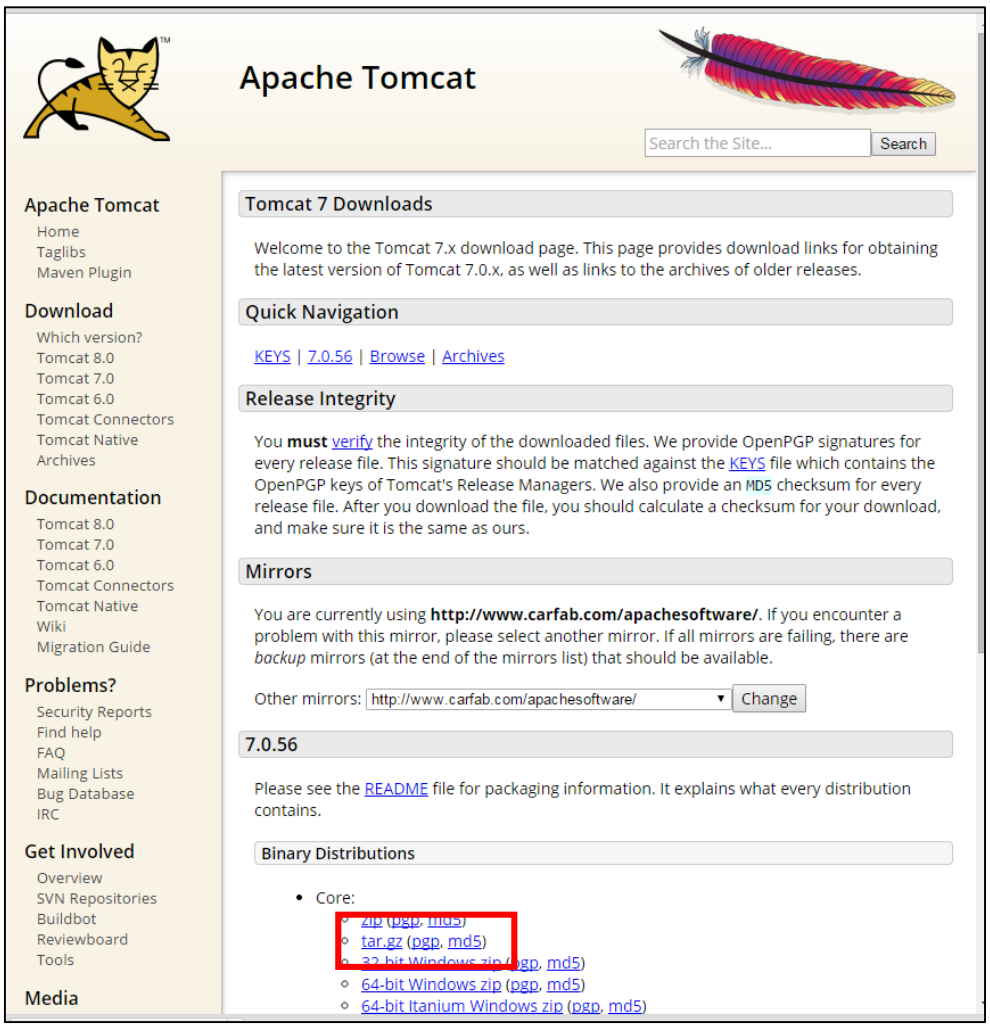


図 75：Tomcat ダウンロードページ

ここでは説明のため、Tomcat 7.0.56 を利用します。ダウンロードが完了したら、WinSCP を使って apache-tomcat-7.0.56.tar.gz ファイルを Raspberry pi にコピーします。

⑥　次に PuTTY を起動します。

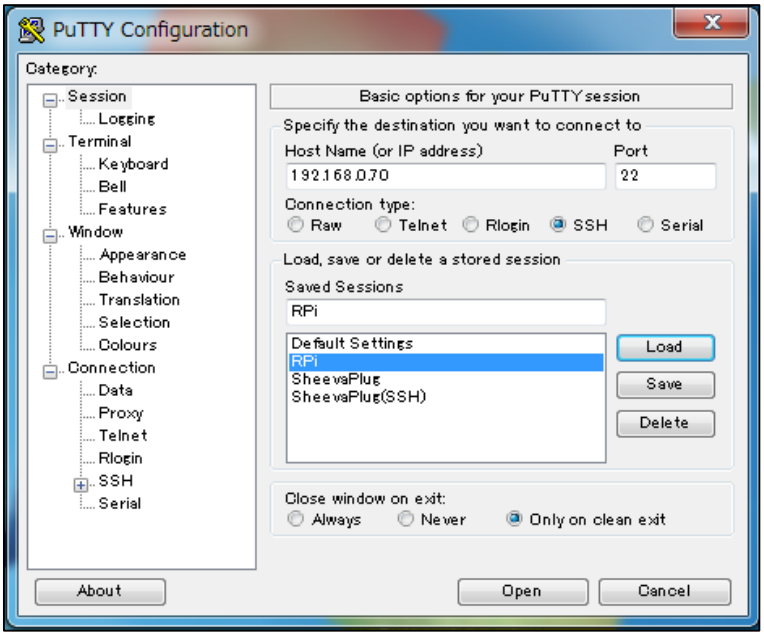


図　76：PuTTY コンフィグ画面

コンフィグ画面で、上図のように設定します。Host Name はこれまで設定した内容を入力して下さい。

⑦　PuTTY で Raspberry Pi にアクセスします。以降のコマンドは PuTTY から実行します。

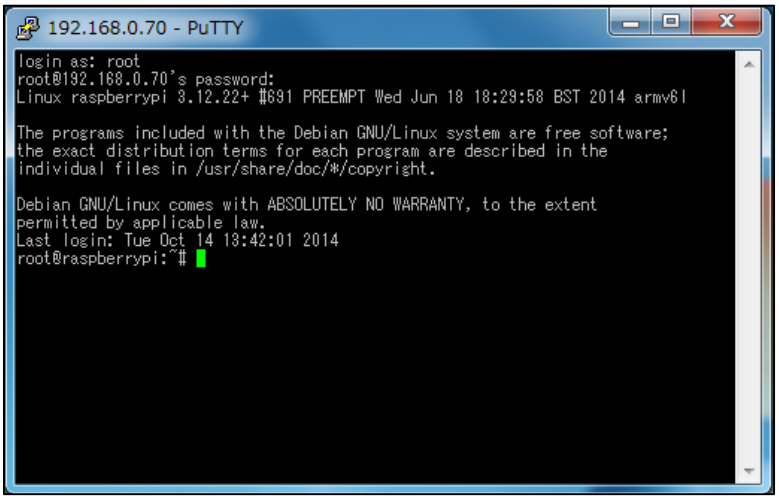


図　77：PuTTY コマンド入力画面

⑧　ルートフォルダに移動します。

```
root@raspberrypi:~# cd ../
```

⑨　Raspberry pi 側でコピーされたファイルを下記のように解凍します。

```
root@raspberrypi:/# tar xvzf apache-tomcat-7.0.56.tar.gz
```

⑩　ファイルの解凍後、Tomcat のディレクトリが作成されますのでディレクトリごと下記のパス（/opt 以下）にコピーしてください。

```
root@raspberrypi:/# mv apache-tomcat-7.0.56/ /opt/apache-tomcat
```

⑪　コピー後、下記の通りに入力し、環境設定ファイルを新規作成します。

```
root@raspberrypi:/# vi /opt/apache-tomcat/bin/setenv.sh
```

⑫　環境設定ファイルには下記の内容を記述してください。

```
#!/bin/sh

export CATALINA_OPTS="-Duser.language=ja -Duser.country=JP -Xms256M
-Xmx384M -XX:MaxPermSize=128M"
```

⑬　ファイルの保存後、下記の通りに入力して環境設定ファイルの権限を変更します。

```
root@raspberrypi:/# chmod 755 /opt/apache-tomcat/bin/setenv.sh
```

⑭　ブラウザからアクセスする用にポート番号をデフォルトの 8080 番から 80 番に変更します。
まず、下記の通りに入力し、設定ファイルを開いてください。

```
root@raspberrypi:/# vi /opt/apache-tomcat/conf/server.xml
```

⑮　下記のように port="8080"と設定されている部分を port="80"に変更します。

```
～～～～～省略～～～～～
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
～～～～～省略～～～～～
```

⑯　デフォルトのログ設定では長期運用を行うと、ログファイルが肥大化し、ディスク容量を圧迫する恐れがあります。簡易的な対策として、既存のログ設定ファイルをリネームし、ログ出力を停止させます。（ここでは改行が入っていますが、全て 1 行で入力してください）

```
root@raspberrypi:/# mv /opt/apache-tomcat/conf/logging.properties
/opt/apache-tomcat/conf/logging.properties.org
```

5. 1. 9. 各種スクリプトの転送と登録

ここでは UECS-Pi アプリケーションの自動起動用スクリプトや IP アドレスをアプリケーション上から変更可能とするためのシェルスクリプトを Raspberry Pi にインストールします。scripts.tar.gz を WinSCP 等で Raspberry Pi にコピーします。scripts.tar.gz は、release フォルダに入っています。

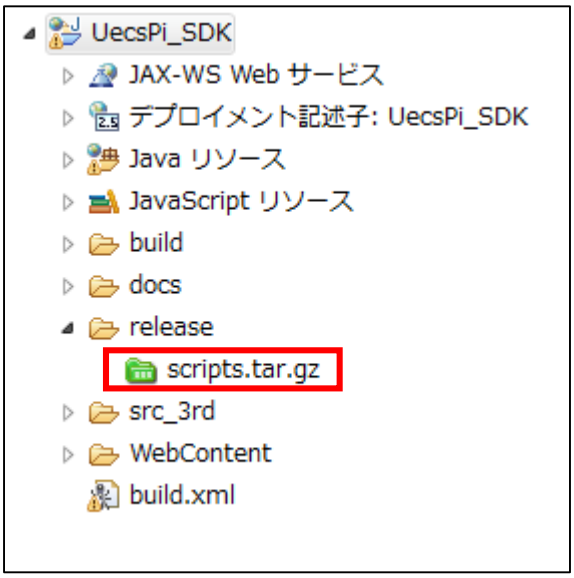


図 78：script.tar.gz の位置

① 下記の通りに入力し、スクリプトファイルを展開し、opt 以下に保存します。

```
root@raspberrypi:/# tar zxvf scripts.tar.gz
root@raspberrypi:/# mv scripts /opt/
```

② ファイルの保存後、下記の通りに入力してファイルの権限を変更します。

```
root@raspberrypi:/# chmod 755 /opt/scripts/*.sh
```

③ 次に、Tomcat 自動起動スクリプトが実行されるよう cron にスクリプトの登録を行います。下記の通りに入力し、設定ファイルを開きます。

```
root@raspberrypi:/# crontab -e
```

④ 1 分置きに起動スクリプト（process_check.sh）が実行されるよう、下記の内容を登録してください。

```
*/1 * * * * /opt/scripts/process_check.sh
```

⑤ WinSCP から scripts.tar.gz ファイルを削除します。

起動と終了

起動スクリプトを cron に登録した場合、Tomcat は自動で起動するようになりますが、手動で Tomcat を起動、または終了させたい場合は下記のスクリプトを実行してください。

① 起動したい場合

```
root@raspberrypi:/# /opt/apache-tomcat/bin/startup.sh
```

② 終了したい場合

```
root@raspberrypi:/# /opt/apache-tomcat/bin/shutdown.sh
```

5.2. アプリケーションインストール

5.2.1. WAR ファイルファイルの作成

① Raspberry Pi のセットアップが完了し、SD カード、AC アダプタ、ネットワークケーブルが接続されているのを確認します。

② WinSCP を起動し、Raspberry Pi に接続します。

③ Eclipse を起動します。

④ UECS-Pi SDK に含まれている web.xml のコメントアウトを以下のように外します。

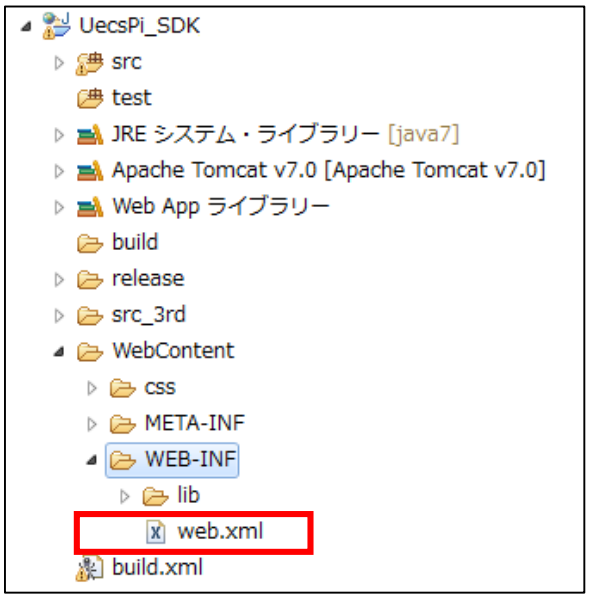


図 79：web.xml の場所

```
～～～～～省略～～～～～
<!--          ←ここを削除する
        <init-param>
            <param-name>configuration</param-name>
            <param-value>deployment</param-value>
        </init-param>
 -->          ←ここを削除する
～～～～～省略～～～～～
```

図 80：コメントアウトを外す箇所

⑤　UECS-Pi SDK に含まれている build.xml を右クリックし「実行」→「2　And ビルド」をクリックします。

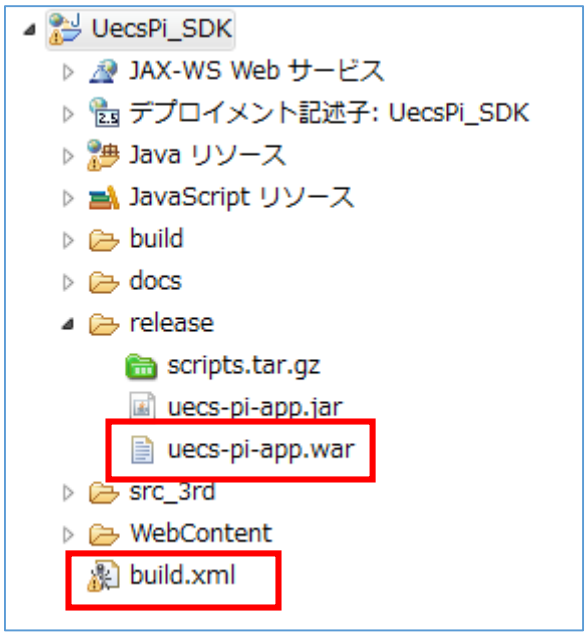


図　81：build.xml、uecs-pi-app.war の場所

⑥　uecs-pi-app.war が新しく作成されるので、WinSCP で/root/フォルダにコピーします。

5. 2. 2. 実機インストール

① PuTTY を起動し、Raspberry Pi に接続します。

② 下記の通りに入力し、uecs-pi-app.war ファイルを Tomcat の webapps フォルダに移動します。

```
root@raspberrypi:/# mv /root/uecs-pi-app.war /opt/apache-tomcat/webapps/
```

③ Tomcat を終了→起動し、war ファイルを読み込ませて立ち上げます。

```
root@raspberrypi:/# /opt/apache-tomcat/bin/shutdown.sh
root@raspberrypi:/# /opt/script/process_check.sh
```

④ ブラウザ（Internet Explorer 等）から Raspberry Pi に設定した IP アドレスの URL(http://[ノード IP アドレス]/uecs-pi-app/)に接続します。

⑤ ログイン画面が表示されるので、管理パスワードに **admin** と入力してログインして下さい。
※UECS-Pi SDK ログインパスワードは、ログイン後にノード設定画面で変更可能です。

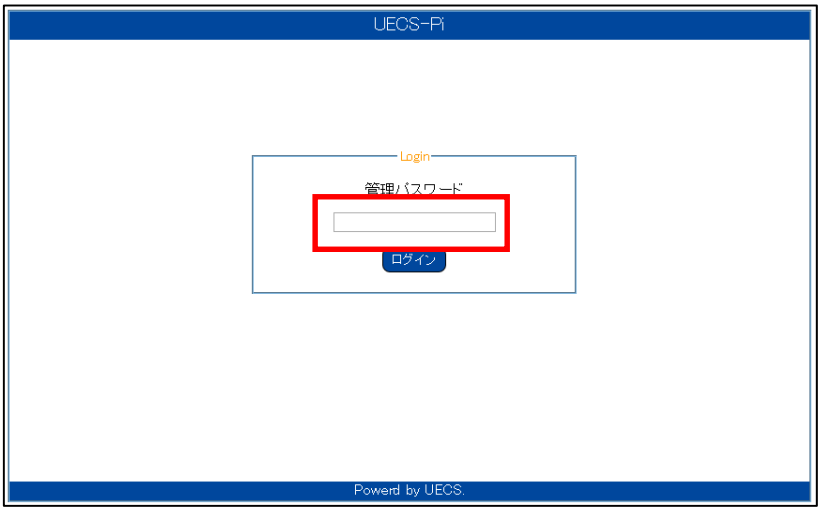


図 82：ログインページ

⑥ ログイン後、トップページが表示されたらセットアップは成功です。

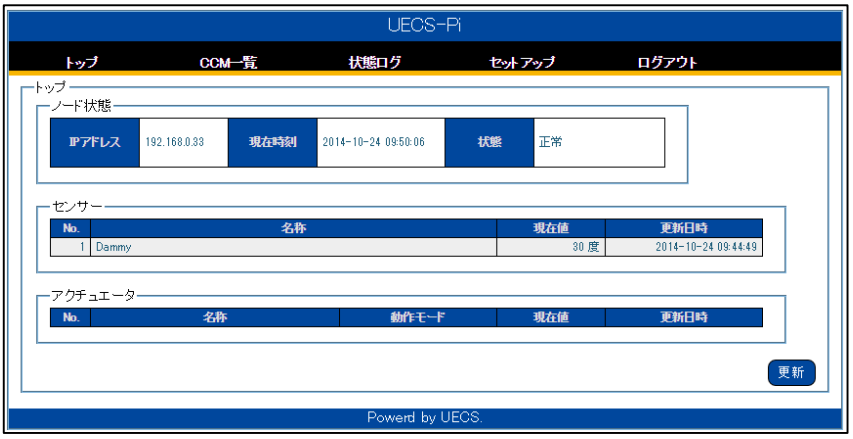


図 83：トップページ

5.2.3. web.xml について

UECS-Pi SDK に含まれる Web アプリケーション設定ファイル(web.xml)内には、アプリケーション動作モードの定義が含まれており、通常はコメントアウトがされています。

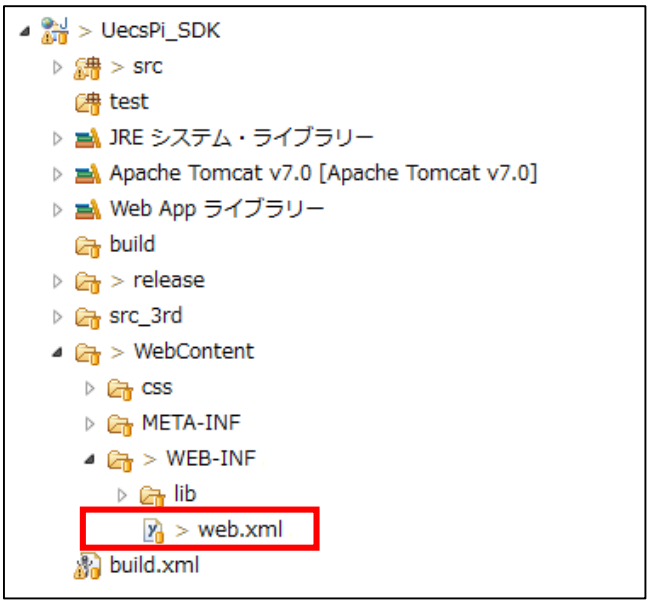


図 84：web.xml の場所

```
～～～～～省略～～～～～
<!--          ←コメントアウトタグ
        <init-param>
            <param-name>configuration</param-name>
            <param-value>deployment</param-value>
        </init-param>
 -->          ←コメントアウトタグ
～～～～～省略～～～～～
```

図 85：コメントアウト箇所

Eclipse で UECS-Pi SDK を動作させる際は、通常ここをコメントアウトしたまま実行して下さい。Wicket の WebUI 処理のエラー情報が詳細に画面表示され、シリアル通信機能や、I2C 通信機能など、開発用 PC ではテストできない機能が OFF となります。また実機インストールして、デバイス通信のテストを行う際は、このコメントアウトを外して下さい。

6. 利用ライセンス

No.	項目	参照 URL
1	Raspbian OS (Linux)	http://www.raspbian.org/
2	Apache Tomcat	http://tomcat.apache.org/
3	Apache Wicket	http://wicket.apache.org/
4	Apache commons	http://commons.apache.org/
5	Pi4J	http://pi4j.com/
6	SQLite JDBC Driver	https://bitbucket.org/xerial/sqlite-jdbc
7	OrmLite	http://ormlite.com/
8	SLF4J	http://www.slf4j.org/
9	Logback	http://logback.qos.ch/

表 22：UECS-Pi SDK の使用オープンソースソフトウェア一覧

7. 免責事項

本製品を使用したことによる一切の損害（一次的、二次的に関わらず）に対し、当社では責任を負いません。

8. お問い合わせ

本製品は無償製品ですので、基本的に当社はサポート保証責任を負いません。ただし、機能追加・品質改善は随時行ってまいりますので、お気づきの点、ご質問、ご要望がございましたら、下記よりお問い合わせください。

（サポートメールアドレス）： support@wa-bit.com